Dhirubhai Ambani Institute of Information Communication Technology

Course : IT314 Software Engineering

Lab VII

Program Inspection, Debugging and Static Analysis

Student Name : Neel Patel

Student ID : 202201494

Group : G32

1. Armstrong Number

```
//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");
	}

Input: 153
Output: 153 is an armstrong Number.
```

I. PROGRAM INSPECTION

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

- **Error 1:** The program assumes that an argument will be provided when running the program (`int num = Integer.parseInt(args[0]);`). This can cause an `ArrayIndexOutOfBoundsException` if no argument is provided. This should be handled with appropriate input validation.
- **Error 2:** The logic inside the `while` loop is incorrect. It should be:

  remainder = num % 10;
  check = check +
  (int)Math.pow(remainder,3);
  num = num / 10;

  This will correctly extract the last digit, calculate its cube, and then remove the last digit.

- **Error 3**: In the output statement, "armstrong" should be capitalized as "Armstrong".

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

- **Category C:** Computation Errors and Category E: Control-Flow Errors are the most relevant for this program. These categories cover issues related to calculations and loops, which are essential aspects of this program.

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

Program inspection can identify logical errors, but it may not be able to catch runtime errors like division by zero if it doesn't occur during the inspection process.

4. Is the program inspection technique worth applicable?

Yes, program inspection is a valuable technique for identifying errors in code, especially for logic and correctness. It can catch many types of errors, but it's important to note that it doesn't replace the need for testing and debugging. It's most effective when used in combination with other testing methods.

II. Code Debugging:

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

- Error 1: In the `while` loop, `remainder` is not calculated correctly. It should be `remainder = num % 10;` instead of `remainder = num / 10;`.

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

- One breakpoint is needed to fix the error.

    a. **What are the steps you have taken to fix the error you identified in the code fragment?**

    Step 1: Change `remainder = num / 10;` to `remainder = num % 10;`.

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

3. Submit your complete executable code?

Here is the corrected code:

```
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; // Corrected line
check = check + (int)Math.pow(remainder,3);
num = num / 10; // Corrected line
}
if (check == n)
System.out.println(n + " is an Armstrong Number");
else
System.out.println(n + " is not an Armstrong Number"); // Corrected line
}
}
```

2. GCD and LCM

```
//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[])
   {
      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();
    }
}
```

Input:4 5
Output: The GCD of two numbers is 1
The GCD of two numbers is 20

I. PROGRAM INSPECTION

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

- **Error 1:** In the `gcd` method, the condition in the `while` loop is incorrect. It should be `while(a % b != 0)` instead of `while(a % b == 0)`.

- **Error 2:** In the `lcm` method, the condition in the `if` statement is incorrect. It should be `if(a% x == 0 && a % y == 0)` instead of `if(a % x != 0 && a % y != 0)`.

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

For this code, Category C: Computation Errors and Category E: Control-Flow Errors are the most relevant. These categories cover issues related to calculations and loops, which are essential aspects of this program.

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

Program inspection can identify logical errors, but it may not be able to catch runtime errors like division by zero if it doesn't occur during the inspection process.

4. Is the program inspection technique worth applicable?

Yes, program inspection is a valuable technique for identifying errors in code, especially for logic and correctness. It can catch many types of errors, but it's important to note that it doesn't replace the need for testing and debugging. It's most effective when used in combination with other testing methods.

II. Code Debugging:

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

- **Error 1**: In the `gcd` method, the condition in the while loop should be `while(a % b != 0)` instead of `while(a % b == 0)`. This is to correctly calculate the greatest common divisor.

- **Error 2**: In the `lcm` method, the condition in the `while` loop should be `if(a % x == 0 && a% y == 0)` instead of `if(a % x != 0 && a % y != 0)`. This is to correctly calculate the least common multiple.

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

- Two breakpoints are needed to fix the errors.

a. What are the steps you have taken to fix the error you identified in the code fragment?

- **Step 1:** Change the condition in the `gcd` method from `while(a % b == 0)` to `while(a % b != 0)`.

  while(a % b != 0) // Corrected line

- **Step 2:** Change the condition in the `lcm` method from `if(a % x != 0 && a % y != 0)` to `if(a% x == 0 && a % y == 0)`.

  if(a % x == 0 && a % y == 0) // Corrected line

3. Submit your complete executable code?

```
while (num > 0) {
remainder = num % 10; // Corrected line
check = check +
(int)Math.pow(remainder,3); num = num /
10; // Corrected line
}
mport 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) // Corrected line
{
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) // Corrected line
return a;
++a;
}
}
public static void main(String args[])
{
Scanner input = new Scanner(System.in);
System.out.println("Enter the two numbers: ");
int x = input.nextInt();
int y = input.nextInt();
System.out.println("The GCD of two numbers is: " + gcd(x,
y)); System.out.println("The LCM of two numbers is: " +
lcm(x, y)); input.close();
}
}
```

3. Knapsack

```
//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]);
      }
   }
}
```

Input: 6, 2000

Output:

Item	Profit	Weight	Take
1	336	784	false
2	674	1583	false
3	763	392	true
4	544	1136	true
5	14	1258	false
6	738	306	true

I. Program Inspection:

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

- Error 1: The variable "N" should be decremented by one to represent the number of items, as array indices typically start from 0. So, change

  int N = Interger.parseInt(args[0]);

  To int N = interger.parseInt(args[0]) – 1;

- Error 2**:**In the "for" loops for generating random instances and calculating the knapsack solution, there is an issue with the increment of the "n" variable. It should be "n++" (post-increment) but is mistakenly used as "n++" (pre-increment). Change:
  int option1 =
  opt[n++][w]; to
  int option1 =
  opt[n][w];
  and
  int option2 = profit[n-2] + opt[n-1]
  [w-weight[n]]; to int option2 = profit[n-1]
  + opt[n-1][w-weight[n]];.

- Error 3: The program prints "Item" as the header for the table, but the actual output has "item" (with a lowercase 'i'). This should be corrected for consistency in the output.

Change:
System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take"); to
System.out.println("Item" + "\t" + "Profit" + "\t" + "Weight" + "\t" + "Take");.

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

- Category C: Computation Errors and Category H: Other Checks would be more effective. The primary issues in the code are related to computation, such as array indexing and variable manipulation. Additionally, there are issues related to output formatting (Category G: Input/Output Errors) that need attention.

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

Program inspection can identify logical and syntactical errors, but it may not detect issues related to incorrect business logic or incorrect mathematical algorithms. For example, it may not determine whether the knapsack problem is solved correctly according to the problem's requirements.

4. Is the program inspection technique is worth applicable?

Yes, program inspection is a valuable technique for identifying and fixing errors in software. In this case, it helped identify various issues, including array indexing errors and output formatting problems. However, it should be combined with testing and debugging to ensure the correctness of the algorithm and the logical flow of the program.

II. Code Debugging

1. **There are several errors in the program:**
   - **Error-1:** Incorrect loop control variable: The loop control variable n is incremented inside the inner loop, which leads to incorrect indexing and iteration.
   - **Error-2:** Incorrect logic in the inner loop: The logic for calculating option1 and option2 is incorrect.
   - **Error-3:** Array index out of bounds: Array indices n and weight[n] go out of bounds, which can lead to runtime errors.
   - **Error-4:** Incorrect initialization of the opt array: The opt array should be initialized with zeros.

2. **How many breakpoints you need to fix those errors?**
   - To fix the errors in the code, you will need at least two breakpoints:
   - One breakpoint to examine the values of n and w within the inner loops.
   - Another breakpoint to check the values of option1 and option2 to understand the logic errors.

   a. **What are the steps you have taken to fix the error you identified in the code fragment?**

```
int option1 = opt[n][w];

if (weight[n] > w) option2 = profit[n] + opt[n-1][w-weight[n]];
opt[n][w] = Math.max(option1, option2);

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

The loop conditions for n and w should be modified to run from 0 to N and 0 to W, respectively.

3. **Submit your complete executable code?**

```
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 = 0; n <= N;
       n++) { for (int w = 0;
       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 the 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]);
}
}
}
```

4. Magic Number

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

Input: Enter the number to be checked 119
Output 119 is a Magic Number.
Input: Enter the number to be checked 199
Output 199 is not a Magic Number.

I. Program Inspection

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

- **Error 1:** There is a logical error in the loop condition. In the inner while loop, it checks if sum is equal to 0. This should be checking if sum is greater than 0 instead, as the loop should continue until sum is no longer a two-digit number.

Change
while(sum==0)
to while(sum>9).

- **Error 2:**There is a missing semicolon after the statements in the inner while loop. Add a semicolon to the end of sum=sum%10 and s=s*(sum/10).

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

Category C: Computation Errors and Category H: Other Checks would be more effective. The primary issues in the code are related to computation and logic. There is a clear logical error in the code that affects the program's functionality.

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

Program inspection can identify logical errors and syntax issues, but it may not detect issues related to mathematical correctness or incorrect algorithm selection for solving a specific problem. In this case, it may not determine if the code correctly identifies magic numbers according to the problem's requirements.

4. Is the program inspection technique is worth applicable?

Yes, program inspection is a valuable technique for identifying and fixing errors in software. In this case, it helped identify issues with logic and syntax. However, it should be combined with testing and debugging to ensure the correctness of the magic number identification algorithm.

II. Code Debugging

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

- Error-1: In the inner while loop condition, it should check if sum is greater than 0, but it checks if sum is equal to 0.
- Error-2:There are missing semicolons at the end of sum=sum%10 and s=s*(sum/10).

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

We will need at least two breakpoints:

1. One at the start of the inner while loop to examine the values of s and sum.

   Another breakpoint at the end of the inner while loop to check if the loop exits correctly.

a. **What are the steps you have taken to fix the error you identified in the code fragment?**

- Inner while loop condition: while(sum==0) should be while(sum>0) to ensure that the digits are processed properly.
- Sum calculation: s=s*(sum/10) needs to be modified to add the digits instead of multiplying, and the logic to reduce the number should also be changed.
- Missing semicolon: There's a missing semicolon at the end of sum=sum%10.

3. **Submit your complete executable 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 num = n, sum;

        while (num > 9) { // Continue until the number becomes a single digit
            sum = 0; // Reset sum for the next iteration
            while (num > 0) { // Sum the digits
                sum += num % 10;
                num /= 10;
            }
            num = sum; // Set num to the sum of digits
        }

        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
```

```
            System.out.println(n + " is not a Magic Number.");
        }
    }
}
```

5. Merge Sort

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

import java.util.*;

public class MergeSort {
    public static void main(String[] args) {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " + Arrays.toString(list));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    // post: array is in sorted (nondecreasing) order
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            // split array into two halves
            int[] 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++) {
            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++;
            }
        }
    }
}

Input: before 14 32 67 76 23 41 58 85
       after 14 23 32 41 58 67 76 85
```

I. PROGRAM INSPECTION:

1. **There are several errors in the provided Merge Sort program:**
    - Missing `import java.util.Arrays;` statement.
    - Incorrect usage of the `leftHalf` and `rightHalf` methods, including incorrect array size calculations.
    - Incorrect usage of the `merge` method with wrongly incremented and decremented variables.
    - Incorrect loop in the `rightHalf` method when copying elements to the `right` array.
    - Some comments mention `second working version` but don't provide sufficient context.

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

The effective category of program inspection for this code would be "Data Reference Errors" and "Computation Errors." These categories encompass most of the errors found in the code, including issues related to array manipulation and calculation errors.

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

The code inspection, as described in the categories, should be able to identify various types of errors. However, this code inspection might not detect issues related to performance, efficiency, or potential logical errors, which might require more in-depth testing or code review.

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

Yes, the program inspection technique is worth applying to this code, as it helps identify a variety of errors related to data reference, computation, and other types of issues. However, it's important to complement program inspection with testing to ensure the code functions correctly and efficiently.

II. Code Debugging :

1. **There are multiple errors in the program. Here are the errors identified:**
   - In the `mergeSort` method, it attempts to split the array and sort it recursively using `leftHalf` and `rightHalf` methods. However, these methods are not implemented correctly.
   - In the `merge` method, it tries to merge the sorted halves, but the parameters `left` and `right` should not be incremented or decremented within the `merge` method.

2. **To fix these errors, you would need multiple breakpoints to address each of the issues identified. Specifically, you would need to:**
   - Fix the implementation of the `leftHalf` and `rightHalf` methods.
   - Correct the way the parameters are passed to `merge` method.

   a. **The steps to fix the errors in the code fragment:**
      - Rewrite the `leftHalf` method to correctly split the array into the left half.
      - Rewrite the `rightHalf` method to correctly split the array into the right half.
      - Modify the `mergeSort` method to correctly call `leftHalf` and `rightHalf` and merge the sorted halves.
      - In the `merge` method, remove the increment and decrement operations of `left` and ‘right’

3. **Submit your complete executable 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); 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 / 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; // 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++;
}
}
}
}
```

6. Multiply Matrics

```
//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");
        }
      }
    }
}
```

Input: Enter the number of rows and columns of first matrix
2 2
Enter the elements of first matrix
1 2 3 4
Enter the number of rows and columns of first matrix
2 2
Enter the elements of first matrix
1 0 1 0
Output: Product of entered matrices:
3 0
7 0

I. Program Inspection:

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

- Error 1:The array indexing in the matrix multiplication loops is incorrect. In the code, the indexing is performed with -1, which is incorrect.

Change:
first[c-1][c-k]*second[k-1][k-d] to
first[c][k]*second[k][d].

- Error 2:There is an issue in the loop boundaries for matrix multiplication. The loops for "c" and "d" should go up to "m" and "q," respectively.

Change:
for (c = 0 ; c < m ; c++) to
for (c = 0 ; c < m ; c++)
and
for (d = 0 ; d < q ; d++) to
for (d = 0 ; d < q ; d++).

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

In this case, Category C: Computation Errors and Category H: Other Checks would be more effective. The primary issues in the code are related to computation, specifically matrix multiplication. Additionally, there are issues with loop boundaries that need to be corrected (Category E: Control-Flow Errors).

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

Program inspection can identify syntax and logical errors, but it may not detect issues related to the mathematical correctness of the matrix multiplication. It may not determine if the multiplication algorithm itself is correct for the given problem requirements.

4. Is the program inspection technique is worth applicable?

Yes, program inspection is a valuable technique for identifying and fixing errors in software. In this case, it helped identify issues with array indexing and loop boundaries. However, it should be combined with testing and debugging to ensure the correctness of the matrix multiplication algorithm.

II. Code Debugging

1. How many errors are there in the program? Mention the errors you have identified.
   - Error-1: Incorrect loop index in matrix multiplication (inside the nested loops).
   - Error-2: The array "multiply" is not properly initialized.
   - Error-3: The sum variable should be reset for each new element in the result matrix.

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

   a. What are the steps you have taken to fix the error you identified in the code fragment?

      To fix these errors, you will need to set breakpoints at the following locations in the code:
      - Inside the nested loop where you perform matrix multiplication.
      - After initializing the "multiply" array.
      - Inside the loop where you calculate the product and reset the "sum" variable.

3. **Submit your complete executable 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(); // Rows of the first matrix
    n = in.nextInt(); // Columns of the first matrix

    int first[][] = new int[m][n]; // First matrix

    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(); // Rows of the second matrix
    q = in.nextInt(); // Columns of the second matrix
```

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

      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();

      // Matrix multiplication logic
      for (c = 0; c < m; c++) {
        for (d = 0; d < q; d++) {
          for (k = 0; k < n; k++) {
            sum += first[c][k] * second[k][d]; // Corrected the index logic
          }
          multiply[c][d] = sum;
          sum = 0; // Reset sum for the next element
        }
      }

      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.print("\n");
      }
    }
  }
}
```

7. Quardatic Probing

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

import java.util.Scanner;

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

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

    /** Function to clear hash table **/
    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;
    }

    /** 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');
      }
  }
```

Input:

Hash table test

Enter size: 5
Hash Table Operations
1. Insert
2. Remove
3. Get
4. Clear
5. Size

1

```
Enter key and value
c computer
d desktop
h harddrive

Output:
Hash Table:
c computer
d desktop
h harddrive
```

I. PROGRAM INSPECTION:

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

- **Error 1:** In the `insert` method, `i += (i + h / h--) % maxSize;` is incorrect. It should be `i += (h* h++) % maxSize;`.
- **Error 2:** In the `remove` method, `i = (i + h * h++) % maxSize;` should be moved inside the for loop, right before the rehashing loop. Otherwise, it will cause an incorrect rehashing and may not remove the key correctly.

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

Category E: Control-Flow Errors and Category G: Input/Output Errors are the most relevant for this program. These categories cover issues related to control flow and handling of input/output.

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

Program inspection can identify logical errors and control flow issues. However, it may not be able to catch runtime errors like null pointer exceptions if they don't occur during the inspection process.

4. Is the program inspection technique worth applicable?

Yes, program inspection is a valuable technique for identifying errors in code, especially for logic and correctness. It can catch many types of errors, but it's important to note that it doesn't replace the need for testing and debugging. It's most effective when used in combination with other testing methods.

II. Code Debugging:

1. **How many errors are there in the program? Mention the errors you have identified.**
    - There don't appear to be any syntactical errors in the provided code.

2. **How many breakpoints you need to fix those errors?**
    - No breakpoints are needed.

3. **Submit your complete executable code?**
    - The provided code is correct and executable.

8. Sorting Array

```
// 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();
        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]);
    }
}
```

```
Input: Enter no. of elements you want in array: 5
       Enter all elements:
       1 12 2 9 7
       1 2 7 9 12
```

I. PROGRAM INSPECTION:

1. **There are several errors in the provided Merge Sort program:**
    - Missing `import java.util.Arrays;` statement.
    - Incorrect usage of the `leftHalf` and `rightHalf` methods, including incorrect array size calculations.
    - Incorrect usage of the `merge` method with wrongly incremented and decremented variables.
    - Incorrect loop in the `rightHalf` method when copying elements to the `right` array.
    - Some comments mention `second working version` but don't provide sufficient context.

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

    The effective category of program inspection for this code would be "Data Reference Errors" and "Computation Errors." These categories encompass most of the errors found in the code, including issues related to array manipulation and calculation errors.

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

    The code inspection, as described in the categories, should be able to identify various types of errors. However, this code inspection might not detect issues related to performance, efficiency, or potential logical errors, which might require more in-depth testing or code review.

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

    Yes, the program inspection technique is worth applying to this code, as it helps identify a variety of errors related to data reference, computation, and other types of issues. However, it's important to complement program inspection with testing to ensure the code functions correctly and efficiently.

II. Code Debugging :

1. **There are multiple errors in the program. Here are the errors identified:**
    - In the `mergeSort` method, it attempts to split the array and sort it recursively using `leftHalf` and `rightHalf` methods. However, these methods are not implemented correctly.
    - In the `merge` method, it tries to merge the sorted halves, but the parameters `left` and `right` should not be incremented or decremented within the `merge` method.

2. **To fix these errors, you would need multiple breakpoints to address each of the issues identified. Specifically, you would need to:**
    - Fix the implementation of the `leftHalf` and `rightHalf` methods.
    - Correct the way the parameters are passed to `merge` method.

    a. The steps to fix the errors in the code fragment:
    - Rewrite the `leftHalf` method to correctly split the array into the left half.
    - Rewrite the `rightHalf` method to correctly split the array into the right half.

- Modify the `mergeSort` method to correctly call `leftHalf` and `rightHalf` and merge the sorted halves.
- In the `merge` method, remove the increment and decrement operations of `left` and `right`.

3. Submit your complete executable 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 you want in the array: ");
        n = s.nextInt();

        int a[] = new int[n]; // Initialize the array

        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }

        // Sorting logic for ascending order
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Swap if a[i] is greater than a[j]
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }

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

9. Stack Implementation

```
//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;
        }
    }

    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();
    }
}


output: 10
            1
            50
            20
            90

            10
```

I. PROGRAM INSPECTION:

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

- Error 1: In the `push` method, the increment of `top` should be `top++;` instead of `top--;`. This way, it correctly updates the top index and pushes the value onto the stack.

- Error 2: In the `display` method, the condition in the for loop should be `i <= top` instead of `i > top`.

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

Category E: Control-Flow Errors and Category D: Comparison Errors are the most relevant for this program. These categories cover issues related to control flow and comparison operations.

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

Program inspection can identify logical errors and control flow issues. However, it may not be able to catch runtime errors like null pointer exceptions if they don't occur during the inspection process.

4. Is the program inspection technique worth applicable?

Yes, program inspection is a valuable technique for identifying errors in code, especially for logic and correctness. It can catch many types of errors, but it's important to note that it doesn't replace the need for testing and debugging. It's most effective when used in combination with other testing methods.

II. Code Debugging:

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

- There are two errors in the `display` method:
- The condition in the `for` loop should be `i < top` instead of `i > top`.
- The loop should increment `i` using `i++`.

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

- You need to fix two breakpoints. These are points in the code where you'll pause the program during debugging to examine the state and behavior of the program.

- The two breakpoints are:
  - The condition in the `for` loop of the `display` method.
  - The loop increment in the `for` loop of the `display` method.

- By setting breakpoints at these points, you can inspect the variables and step through the code to understand how it's behaving.

a. What are the steps you have taken to fix the error you identified in the code fragment?

- I've identified the errors in the `display` method, which involve incorrect loop conditions and increments.

- Specifically:
  - I changed the condition in the `for` loop from `i > top` to `i <= top` to ensure that the loop iterates over the correct range of elements in the stack.
  - I also changed the loop increment from `i--` to `i++` to properly iterate through the elements.

  - By making these corrections, the `display` method will now correctly print the elements of the stack.

3. Submit your complete executable code?

```
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
}
```

```
}

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

public void display() {
for (int i = 0; i <= top; i++) {
System.out.print(stack[i] + " ");
}
System.out.println();
}
}
```

10. Tower of Hanoi

```
//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)
      }
   }
}

Output: Disk 1 from A to C
            Disk 2 from A to B
            Disk 1 from C to B
            Disk 3 from A to C
            Disk 1 from B to A
            Disk 2 from B to C
            Disk 1 from A to C
```

I. Program Inspection:

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

- **Error 1:** In the `doTowers` method, the recursive call is incorrect. Instead of `doTowers(topN++, inter--, from+1, to+1)`, it should be `doTowers(topN - 1, inter, from, to)`.

- **Error 2:** In the `doTowers` method, the parameters `inter`, `from`, and `to` should be of type`char` and cannot be manipulated like numbers. So, the recursive call should be `doTowers(topN - 1, inter, from, to)`.

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

Category E: Control-Flow Errors and Category D: Comparison Errors are the most relevant for this program. These categories cover issues related to control flow and comparison operations.

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

Program inspection can identify logical errors and control flow issues. However, it may not be able to catch runtime errors like null pointer exceptions if they don't occur during the inspection process.

4. Is the program inspection technique worth applicable?

Yes, program inspection is a valuable technique for identifying errors in code, especially for logic and correctness. It can catch many types of errors, but it's important to note that it doesn't replace the need for testing and debugging. It's most effective when used in combination with other testing methods.

II. Code Debugging:

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

- There are two errors in the program:

1. The increment and decrement operators in the recursive calls inside the `doTowers` method.
2. The parameters for the recursive calls should be rearranged.

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

- You need to fix two breakpoints. These are points in the code where you'll pause the program during debugging to examine the state and behavior of the program.

- The two breakpoints are:
  - The recursive call to `doTowers` for the first recursive step.
  - The recursive call to `doTowers` for the second recursive step.

- By setting breakpoints at these points, you can inspect the variables and step through the code to understand how it's behaving.

a. What are the steps you have taken to fix the error you identified in the code fragment?

- I've identified the errors in the recursive calls inside the `doTowers` method.

- Specifically:
  - I changed `doTowers(topN++, inter--, from+1, to+1)` to `doTowers(topN - 1, from, to, inter)`. This fixes the issue with incrementing `topN` and decrementing `inter` incorrectly.
  - I also rearranged the parameters to match the correct order of `topN`, `from`, `inter`, and `to`.

- By making these corrections, the `doTowers` method will now correctly solve the Tower of Hanoi problem.

3. Submit your complete executable 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);
}
}
}
```