Lab-7
Program Inspection, Debugging and Static Analysis

Manthan Parmar 202201416

For the parts of Program Inspection and Static Analysis I have used the following code from https://github.com/intrepidcoder/monopoly/blob/master/monopoly.js
It is the code for the monopoly board game, it has 2343 LOC and is written in JavaScript.

Program Inspection

Category A: Data Reference Errors

- **Uninitialized variable p in this.next() function**
  - p is referenced without being properly initialized. It seems intended to hold player data but without assignment, could lead to runtime error.
- **Implicit declaration of index in the auction() function**
  - index is used without var, let, or const, which can unintentionally make it global, leading to potential bugs.

Category B: Data Declaration Errors

- **Implicit Globals**
  - Variables such as sq and p are used across multiple functions without explicitly declaring them. This may cause scope-related issues, making the code hard to debug.
- **Missing Declarations**
  - index should be declared properly to avoid leaking into global scope.

Category C: Computation Errors

- No risky computations, like division or floating point operations, that could cause precision or overflow errors.

Category D: Comparison Errors

- No issue with logical expression handling and comparisons.

Category E: Control Flow Errors

- **Potential Infinite Loop**
  - while(true) loop could run indefinitely if the required conditions for exit are not met.
- **Off-by-One Error**
  - If pcount is 0 or undefined, the code adjusting currentbidder might fail, causing game logic to break.

Category F: Interface Errors

- No mismatch between function parameters and arguments. All functions seem to exchange information correctly.

Category G: Input/Output Errors

- **HTML Markup Error**
  - Missing </div> tag in popup() function, may result in UI rendering issues.
- **Error handling for User Input**
  - Function does not validate bid inputs. It should handle cases where bids are empty or negative to avoid incorrect behavior.

Category H: Other Checks

- **Global Variables Referenced only once**
  - Some variables, like highestbidder, are initialized but not effectively used, suggesting redundancy or dead code.
- **Warning Review**
  - Missing declarations and implicit globals would likely trigger warnings during static analysis or compilation.

Summary of Program Inspection:

Category	Errors Found	Explanation
Data Reference Errors	Yes	Uninitialized variables (p) and global

		leaks (index).
Data Declaration Errors	Yes	Implicit globals may cause scope-related issues.
Computation Errors	No	No risky computations identified.
Comparison Errors	No	Logical comparisons are correctly implemented.
Control Flow Errors	Yes	Infinite loop potential and off-by-one error detected.
Interface Errors	No	Parameter and argument usage is consistent.
Input/Output Errors	Yes	HTML issues and lack of input validation.
Other Checks	Yes	Some global variables seem redundant or underused.

Code Debugging

PROGRAM 1 Sorting Array

Original Code

```
// 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
```

How many errors are there in the program?

There are 4 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Class Name:** Ascending _Order has a space character in between.
    - No breakpoint needed, Syntax Error.
- **Loop Condition:** i>=n is incorrect.
    - Set breakpoint in loop.
- **Sort Logic:** a[i]<=a[j] incorrect for ascending order.
    - Set breakpoint to check swapping mechanism.
- **Object Closure:** Scanner object s.
    - No breakpoint needed.

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

- Changed Ascending _Order to AscendingOrder.
- Changed i>=n to i<n and removed ; at end.
- Changed a[i]<=a[j] to a[i]>a[j].
- Add s.close(); at end of code.

Complete Executable Code

```
 // Sorting the array in ascending order
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 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();
        }

        // Sorting logic
        for (int i = 0; i < n; i++) {  // Corrected condition
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {  // Corrected comparison operator
                    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]);

        s.close();  // Closing the Scanner object
    }
}
```

PROGRAM 2 Magic Number

Original Code

```
 // 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.
```

How many errors are there in the program?

There are 4 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Sum Logic:** s = s*(sum/10); is incorrect.

    - Set breakpoint inside the inner while loop to check the value of s after each iteration.
- **Infinite Loop:** while (sum==0).
    - Set breakpoint to confirm loop work correctly.
- **Missing Semicolon :** sum = sum%10
    - No breakpoint needed, syntax error.
- **Object Closure:** Scanner object ob.
    - No breakpoint needed.

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

- Changed s = s*(sum/10); to s = s + (sum%10);
- Changed while(sum==0) to while(sum>0)
- Add ; at end of sum = sum%10;.
- Add ob.close(); at end of code.

Complete Executable Code

```java
 // Program to check if a number is a Magic number in Java
import java.util.Scanner;

public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;

        // Main logic to check if the number is a magic number
        while (num > 9) {
            sum = num;
            int s = 0;

            // Corrected the inner loop condition
            while (sum > 0) {
                s = s + (sum % 10);  // Corrected the summation logic
                sum = sum / 10;      // Extract the next digit
            }
            num = s;
        }

        // Output based on final result
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }

        // Close the Scanner object
        ob.close();
    }
}
```

PROGRAM 3 Merge Sort

Original Code

```
 // 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
```

How many errors are there in the program?

There are 3 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Array Slicing:** leftHalf(array + 1) and rightHalf(array - 1) incorrect since array is array reference.
  - Breakpoint inside mergeSort method.
- **Merge Logic:** merge(array,left + +,right - -);
  - Set breakpoint here to confirm correct merge
- **Out of Bounds :** Not accounted for odd size arrays.
  - Set breakpoint to verify left and right array values.

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

- Use Arrays.copyOfRange():

```
int[] left = Arrays.copyOfRange(array,0,array.length/2);

int[] right =  Arrays.copyOfRange(array,array.length/2,array.length);
```

- Changed merge(array,left++,right–); to merge(array,left,right);
- int size1 = array.length/2 to int size1 = (array.length + 1)/2;

Complete Executable Code

```
 // 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 using Arrays.copyOfRange
            int[] left = Arrays.copyOfRange(array, 0, (array.length + 1) / 2);
            int[] right = Arrays.copyOfRange(array, (array.length + 1) / 2,
array.length);

            // 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 + 1) / 2; // Adjusted for odd-sized arrays
        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.
    // 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++;
            }
        }
    }
}
```

PROGRAM 4 Tower of Hanoi

Original Code

```
 //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
```

How many errors are there in the program?

There are 2 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Post Operators:** doTowers(topN++,inter–,from+1,to+1)
    - Set breakpoint at method call to verify values passed.
- **Addition of characters:** from+1 and to+1 treat as integers.
    - Set breakpoint to check character values passed.

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

- Changed doTowers(topN++,inter–,from+1,to+1) to doTowers(topN-1,from,to,inter);
- Remove ++ and – on topN and inter in recursive characters.

Complete Executable Code

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

PROGRAM 5 Armstrong

Original Code

```
 //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.
```

How many errors are there in the program?

There are 3 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Remainder Calculation:** remainder = num/10;
  - Set breakpoint at this point.
- **num update:** num = num%10.
  - Set breakpoint to verify num correctly updated each time until zero.
- **Array Index Out of Bounds:** int num = Integer.parseInt(args[0]);.
  - No breakpoint needed, input validation issue.

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

- Changed remainder = num/10; to remainder = num%10;
- Changed num = num%10 to num = num/10;

- Check for command line arguments before parsing input number to avoid ArrayIndexOutOfBoundsException.

Complete Executable Code

```
// Armstrong Number
class Armstrong {
    public static void main(String args[]) {
        // Check if there is a command-line argument
        if (args.length == 0) {
            System.out.println("Please provide a number as a command-line argument.");
            return;
        }

        // Parse the command-line argument
        int num = Integer.parseInt(args[0]);
        int n = num; // Store original number for comparison
        int check = 0, remainder;

        // Calculate the Armstrong number check value
        while (num > 0) {
            remainder = num % 10;  // Corrected to get the last digit
            check = check + (int) Math.pow(remainder, 3);
            num = num / 10;  // Corrected to remove the last digit
        }

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

PROGRAM 6 Knapsack

Original Code

```
//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
```

How many errors are there in the program?

There are 4 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Increment Loop:** int option1 = opt[n++][w];
  - Set breakpoint to ensure n not modified during array indexing.
- **Access Profit and Weight Arrays:** int option2 = profit[n-2] + opt[n-1][w-weight[n]].
  - Set breakpoint to check values of option1 and option2.
- **Option2 condition:** if (weight[n]>w).
  - No breakpoint needed.
- **Array Index Out of Bounds in Print Loop:** profit[n] and weight[n] from 1 to N.
  - No breakpoint needed.

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

- Changed int option1 = opt[n++][w]; to int option1 = opt[n][w];
- Changed int option2 = profit[n-2] + opt[n-1][w-weight[n]]; to option2 = profit[n] + opt[n-1][w-weight[n]];
- Init option2 to Integer.MIN_VALUE only if weight[n]<=w
- Ensure if N>0.

Complete Executable Code

```
// Knapsack
public class Knapsack {

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Please provide number of items and maximum weight as 
arguments.");
            return;
        }

        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];  // Corrected increment

                // Take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] <= w) {  // Corrected condition
                    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]);
        }
    }
}
```

PROGRAM 7 Stack Implementation

Original Code

```
 //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
```

How many errors are there in the program?

There are 4 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Push Logic:** top - - is incorrect.
  - Place breakpoint after top assignment.
- **Loop Condition:** i>top.
  - Set breakpoint in display method.
- **Pop Logic:** top++;.
  - Set breakpoint to verify the top index updated correctly.
- **Unused Import:** Remove import java.util.Arrays.
  - Not necessary.

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

- Change top–; to top++;
- Changed i>top to i<=top.
- Changed top>-0 and change it top–;

- Remove unnecessary import statements for Arrays.

Complete Executable Code

```
 // Stack implementation in Java

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++;  // Corrected to increment top
            stack[top] = value;
        }
    }

    public void pop() {
        if (!isEmpty()) {
            top--;  // Corrected to decrement 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++) {  // Corrected 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();
    }
}
```

PROGRAM 8 Quadratic Probing

Original Code

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

import java.util.Scanner;

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

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

    /** Function to clear hash table **/
    public void makeEmpty()
    {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

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

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

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

    /** 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
```

How many errors are there in the program?

There are 6 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Operator Syntax:** i = (i+h/h–)% maxSize;
    - Place breakpoint at line.
- **Logic in insert:** h/h – -.
    - Set breakpoint after assignment.
- **Logic in get:** h * h ++
    - Set breakpoint to ensure index calculation.
- **Logic in remove:** keys[i] can be NULL.
    - Breakpoint inside loop.
- **Hash Code Calculation:** ensure not non negative.
    - Set breakpoint to check hash values.
- **Unused import:** Scanner object s.
    - No breakpoint needed.

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

- Changed i += (i + h/h– -) % maxSize to i+= (h * h) % maxSize;
- Update to use h++;
- Changed i = (i + h*h)% maxSize, h++;
- Add check for kys[i] being null before equals()
- hash function = (key.hasCode()%maxSize + maxSize) %maxSize;
- Validate Scanner Usage

Complete Executable Code

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

import java.util.Scanner;

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

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

    /** Function to clear hash table **/
    public void makeEmpty()
    {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

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

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

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

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

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

    /** 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 += (h * h) % maxSize; // Fixed increment logic
            h++; // Increment h for quadratic probing
        } 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; // Fixed increment logic
            h++; // Increment h for quadratic probing
        }
        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 (keys[i] != null && !key.equals(keys[i]))
            i = (i + h * h) % maxSize; // Fix potential null issue

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

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

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

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

            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
                System.out.println("Enter key and value");
                qpht.insert(scan.next(), scan.next());
                break;
            case 2 :
                System.out.println("Enter key");
                qpht.remove(scan.next());
                break;
            case 3 :
```

```java
                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');
    }
}
```

PROGRAM 9 GCD and LCM

Original Code

```
 //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
```

How many errors are there in the program?

There are 3 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **GCD Calculation:** while(a%b==0)
    - Need breakpoint in gcd method.
- **LCM Calculation:** condition return a when neither divides a.
    - Set breakpoint in lcm method.
- **Inefficient LCM:** Logic inefficient.
    - No need.

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

- Changed while condition to while(a%b!=0).
- Changed if (a%x!=0 && a%y!=0) to if(a%x==0 && a%y==0).
- lcm(x,y) = (x*y)/gcd(x,y).

Complete Executable Code

```
 // 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) ? x : y; // a is greater number
        b = (x < y) ? x : y; // b is smaller number

        r = b;
        while(a % b != 0) // Corrected condition
        {
            r = a % b;
            a = b;
            b = r;
        }
        return b; // Return b, which is the GCD
    }

    static int lcm(int x, int y)
    {
        return (x * y) / gcd(x, y); // Efficient calculation using GCD
    }

    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();
    }
}
```

PROGRAM 10 Multiply Matrices

Original Code

```
 //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
```

How many errors are there in the program?

There are 3 errors in the program.

Mention the errors you have identified. How many breakpoints do you need to fix those errors?

- **Access of matrix indices:** first[c-1][c-k] and second[k-1][k-d]
  - Set breakpoint in multiplication loop.
- **Calculation of Sum :** sum reused without resetting for each element in the result matrix.
  - Set breakpoint to ensure sum reset correctly.
- **Output Format of Matrix:** “\t” no proper visual separation.
  - No breakpoint needed.

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

- Changed first[c-1][c-k] and second[k-1][k-d] to first[c][k] and second[k][d].
- Init sum = 0 at beginning of innermost loop.
- Changed output to System.out.printf

Complete Executable Code

```
// Java program to multiply two matrices
import java.util.Scanner;

class MatrixMultiplication
{
    public static void main(String args[])
    {
        int m, n, p, q, sum, 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");
// Corrected prompt
        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();

            // Multiply the matrices
            for (c = 0; c < m; c++)
            {
                for (d = 0; d < q; d++)
                {
                    sum = 0;  // Reset sum for each element in result matrix
                    for (k = 0; k < n; k++) // Corrected inner loop index for
multiplication
                    {
                        sum += first[c][k] * second[k][d]; // Corrected access
                    }
                    multiply[c][d] = sum; // Store the result
                }
            }
            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"); // Printing with tab

                System.out.print("\n");
            }
        }
        in.close(); // Close the scanner
    }
}
```

Static Analysis

For static analysis for JavaScript codebase I have made use of the ESLint tool to identify errors in the code without executing it.

ESLint is chosen since it is widely used for linting JavaScript and provides comprehensive checks for common issues like undefined variables, unused variables and redefined variables.

To configure, I made changes to eslint.config.mjs file.

Specifically I set no-unused-vars (unused variables) and no-undef (undefined variables) to warnings instead of errors since they are giving false positive errors, as $ from jQuery is referencing a global variable in a browser or other environment.

So ESLint might not recognise them unless explicitly declared as globals in ESLint configuration.

The errors that are highlighted by ESLint are stored in eslint_results.txt which is placed in the GitHub folder for Lab 07.

A short Summary of linting results:

- 371 Problems found by the linting tool on the codebase monopoly.js.
- Of which 14 are errors (redeclaration of variables).
- And the rest 357 are false positives which are set as warnings for undefined and unused variables.

The below is code of **eslint.config.mjs** file

```
import globals from "globals";

import pluginJs from "@eslint/js";



export default [

  {

    files: ["**/*.js"],

    languageOptions: {

      sourceType: "commonjs",

      globals: {

        ...globals.browser, // This will spread the browser globals

      },

    },

    rules: {

      'no-unused-vars': 'warn', // Warn on unused variables

      'no-undef': 'warn', // Warn on undefined variables

    },

  },

  pluginJs.configs.recommended,

];
```