IT314
Software Engineering

Lab 7

Name: Jainam Patel
Student Id: 202201514

Q-1. Choose your own code fragment from GitHub in any programming language and do the inspection and debugging.

Code Link:
https://github.com/kdave12/15112-TermProject-2000-lines-of-Python-code/blob/master/termProject.py
Language: Python
Lines Of Code: 1700 Lines

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

Upon reviewing the program using the program inspection categories, I have identified the following errors:

1. **Data Reference Errors (Category A)**:
    - **Uninitialized variables**: Variables like **`data.helpCx`**, **`data.helpCy`**, and **`data.helpRadius`** are used in `finalMousePressed`, but they may not be initialized properly before use.
    - **Array or list access bounds**: No explicit errors regarding list bounds were found, but further checks would be needed where lists or arrays are accessed to confirm no out-of-bound access is possible.
    - **Pointer/reference issues**: The program does not explicitly manage pointers or references, but there are potential errors if tkinter's objects (e.g., **`canvas`** or image files) are not properly initialized.
2. **Data-Declaration Errors (Category B)**:
    - **Missing imports**: **`PhotoImage`** is used without importing the relevant module (likely from **`tkinter`**).

- **Incorrect initialization of data fields**: The **`Struct()`** function is used to initialize `data`, but some fields are missing explicit initialization.

3. **Computation Errors (Category C)**:
   - No specific computation errors were identified, but mixed-type errors could arise if type handling in **`threshold()`** or **`edgeDetection()`** is improper once these functions are implemented.

4. **Comparison Errors (Category D)**:
   - **Mixed-Mode Comparisons:** In the function **`applyThreshold()`**, there is a comparison **`if edge_value > threshold:`** where **`edge_value`** is calculated using **`int()`** and **`threshold`** seems to be a manually input value, possibly an integer or a float. While the comparison between an integer and a float might not directly cause an error in Python, it is something to be cautious about since it could lead to unexpected results in other languages or environments that require explicit type handling.

5. **Control-Flow Errors (Category E)**:
   - **Mouse release handling (`mouseReleaseWrapper`)**: There is a potential issue with passing parameters correctly when calling the **`redrawAllWrapper`** function.
   - **Event-handling robustness**: The code for mouse events may fail if certain states or conditions are not met (e.g., invalid canvas dimensions or uninitialized data variables).
   - **File loading issues**: The hardcoded paths for loading images (**`final =`**

`"C:/Users/krishna/Desktop/112/TermProject/finalSea.gif"`) could cause issues if the paths are invalid.

6. **Interface Errors (Category F)**:
    - There is no mismatch between the number of parameters passed in function calls, but further checks would be required for the correct use of tkinter's built-in functions, especially related to GUI handling.

7. **Input/Output Errors (Category G)**:
    - **File handling**: Hardcoded file paths may lead to runtime errors if the paths don't exist, and there is no exception handling around file access.

8. **Other Checks (Category H):**
    - **Variables never referenced**: The program defines some variables, such as those in the `Struct()` initialization, but not all of them are used in the visible portions of the code. For example, `data.isHelpScreen` and `data.helpCx, data.helpCy` may not be referenced unless they are used in the parts of the code that are not visible here.
    - **Function missing**: The key functions `threshold()` and `edgeDetection()` are not implemented in the provided code, which is a major missing function error. Without these functions, the program will not be able to perform as intended, especially if these are critical to the overall functionality.

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

The most effective categories for inspecting this program are:

- **Category A: Data Reference Errors**: Given that the program manipulates many variables (especially GUI-related) and references objects like canvas, uninitialized or unset variables are a key area to inspect.
- **Category E: Control-Flow Errors**: Since this program is event-driven and relies on mouse events and canvas updates, inspecting control flow errors will ensure proper event handling and program termination.
- **Category G: Input/Output Errors**: The program relies on external files (image assets), and handling those resources correctly is essential.

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

The inspection technique struggles to identify **logical errors** or **domain-specific errors** in the unimplemented functions such as `threshold()` or `edgeDetection()`. Without seeing the logic of these functions or knowing their purpose, it's hard to tell if the program will behave as expected once they are implemented.

Also, the technique does not identify **performance-related errors** or **GUI responsiveness issues**, which are crucial for event-driven GUI programs like this one.

4. Is the program inspection technique worth applying?

Yes, the program inspection technique is worth applying. It helps identify critical issues related to data handling, control flow, and resource management early on. The categories like data reference, control flow, and

input/output errors are highly applicable, as they target the foundational aspects of this tkinter-based program.

By systematically going through each error category, we can catch potential issues before runtime, which reduces debugging effort and increases code reliability. However, this inspection should be followed up with dynamic testing, especially for GUI-related issues and unimplemented functions.

<u>Q-2.</u> CODE DEBUGGING: Debugging is the process of localizing, analyzing, and removing suspected errors in the code (Java code given in the .zip file) Instructions (Use Eclipse/Netbeans IDE, GDB Debugger)

• Open a NEW PROJECT. Select Java/C++ application. Give suitable name to the file.
• Click on the source file in the left panel. Click on NEW in the pull down menu.
• Select main Java/C++ file.
• Build and Run the project.
• Set a toggle breakpoint to halt execution at a certain line or function
• Display values of variables and expressions
• Step through the code one instruction at a time
• Run the program from the start or continue after a break in the execution
• Do a backtrace to see who has called whom to get to where you are
• Quit debugging.

1. Armstrong

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

There are **3 errors** in the code.

1. **Incorrect calculation of the remainder:**
    - The statement `remainder = num / 10;` should be `remainder = num % 10;` because `/` is for division and `%` is used for calculating the remainder.
2. **Incorrect update of `num`:**
    - The statement `num = num % 10;` is wrong because it will continuously keep the last digit instead of reducing the number by dividing by 10. It should be `num = num / 10;`.
3. **Incorrect power calculation:**
    - The number of digits in the input number should be used for power calculation, but in the current code, it is hardcoded to 3, which may only work for 3-digit numbers. You need to calculate the number of digits dynamically for correct power calculation.

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

You would need **3 breakpoints** to fix these errors:

1. Breakpoint to check if the remainder is being calculated correctly.
2. Breakpoint to check if `num` is being updated correctly to remove the last digit.
3. Breakpoint to check if the number of digits is calculated and applied correctly to the `Math.pow()` function.

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

1. **Fix the calculation of remainder:**

    - Change `remainder = num / 10;` to `remainder = num % 10;` to correctly extract the last digit of the number.
2. **Fix the update of `num`:**
    - Change `num = num % 10;` to `num = num / 10;` to correctly reduce the number by removing its last digit.
3. **Calculate the number of digits dynamically:**
    - Add logic to calculate the number of digits before entering the loop. You can use a temporary variable to store the value of `num` and find its length.

3. Submit your complete executable code.

```
// Armstrong Number

class Armstrong {

    public static void main(String args[]) {

        // Parse the number from command-line argument

        int num = Integer.parseInt(args[0]);

        int n = num; // Copy of original number to use later

        int check = 0, remainder;


        // Find the number of digits

        int digits = 0;

        int temp = num; // Temporary variable to calculate the number of digits

        while(temp != 0) {

            temp = temp / 10;

            digits++;

        }
```

```
        // Check if Armstrong number
        while(num > 0) {
            remainder = num % 10; // Extract last digit
            check = check + (int)Math.pow(remainder, digits); // Raise remainder to the power of number of digits
            num = num / 10; // 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");
    }
}
```

2. GCD And LCM

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

There are **3 errors** in the program:

1. **Incorrect while condition in `gcd` function:**
    - The current condition is `while(a % b == 0)`, which would cause an infinite loop when the remainder is 0. It should be `while(a % b != 0)` to correctly calculate the GCD.
2. **Logic error in `lcm` function:**

- The condition `if(a % x != 0 && a % y != 0)` is incorrect. It should be `if(a % x == 0 && a % y == 0)` to find the LCM.

3. **GCD Logic:**
   - The variables a and b are not properly swapped. After finding the remainder, `a = b` and `b = r` need to be swapped as part of the Euclidean algorithm.

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

You would need **3 breakpoints** to fix the errors:

1. Breakpoint in the `gcd` function to check the while loop condition and the value of `a % b`.
2. Breakpoint in the `lcm` function to check the condition when both `x` and `y` divide a evenly.
3. Breakpoint to check the final GCD and LCM values.

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

1. **Fix the while condition in the gcd function:**
   - Change `while(a % b == 0)` to `while(a % b != 0)`.
2. **Fix the LCM logic:**
   - Change `if(a % x != 0 && a % y != 0)` to `if(a % x == 0 && a % y == 0)`.

3. Submit your complete executable code.

```
import java.util.Scanner;
public class GCD_LCM
{
   // Function to calculate GCD using Euclidean algorithm
   static int gcd(int x, int y)
   {
```

```
        int r = 0, a, b;
        a = (x > y) ? x : y; // a is the larger number
        b = (x < y) ? x : y; // b is the smaller number

        while(b != 0)
        {
            r = a % b;
            a = b;
            b = r;
        }
        return a; // GCD is stored in 'a'
    }

    // Function to calculate LCM
    static int lcm(int x, int y)
    {
        return (x * y) / gcd(x, y); // LCM formula: (x * y) / GCD(x, y)
    }

    public static void main(String args[])
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();
        // Output GCD and LCM
        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

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

There are **4 errors** in the program:

1. **Post-increment in `option1` assignment (line 24):**
   - The line `int option1 = opt[n++][w];` increments n by 1 after accessing the array element, which results in skipping items and leading to incorrect results. It should be `int option1 = opt[n][w];` to avoid incrementing n.
2. **Incorrect item selection in `option2` (line 28):**
   - The line `option2 = profit[n-2] + opt[n-1][w-weight[n]];` incorrectly references `profit[n-2]` instead of `profit[n]`. It should be `option2 = profit[n] + opt[n-1][w-weight[n]];` to correctly consider the profit of the current item n.
3. **Weight comparison logic in `option2` (line 27):**
   - The condition `if (weight[n] > w)` should be `if (weight[n] <= w)` because we want to consider items whose weight is less than or equal to the current capacity w.
4. **Random weight generation (line 9):**
   - The current random weight generation formula `weight[n] = (int) (Math.random() * W);` could generate a weight of 0. To ensure the items have non-zero weights, the formula should be updated to `weight[n] = (int) (Math.random() * (W - 1)) + 1;`.

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

You would need **4 breakpoints** to fix the errors:

1. To check the value of `opt[n][w]` when computing `option1`.
2. To check the `profit[n]` and `opt[n-1][w-weight[n]]` when computing `option2`.
3. To verify the condition `if (weight[n] <= w)` when deciding to take the current item.
4. To validate the random generation of weights to ensure no zero values are assigned.

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

1. **Fix the post-increment issue:**
   - Change `int option1 = opt[n++][w];` to `int option1 = opt[n][w];`.
2. **Fix the profit calculation in `option2`:**
   - Change `option2 = profit[n-2] + opt[n-1][w-weight[n]];` to `option2 = profit[n] + opt[n-1][w-weight[n]];`.
3. **Fix the weight comparison:**
   - Change `if (weight[n] > w)` to `if (weight[n] <= w)`.
4. **Fix the random weight generation:**
   - Update `weight[n] = (int) (Math.random() * W);` to `weight[n] = (int) (Math.random() * (W - 1)) + 1;`.

3. 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 - 1)) + 1; // ensure non-zero weights
}

// 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];

// solve knapsack problem
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] + 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]);
      }
   }
}
```

4. Magic Number

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

There are **4 errors** in the original program:

- **Error 1:** The inner `while` loop condition was `while(sum == 0)`, which is incorrect. It should be `while(sum > 0)` to ensure the loop runs while there are digits left to process.
- **Error 2:** The statement `s = s * (sum / 10)` incorrectly multiplies digits instead of adding them. The correct logic should be `s = s + (sum % 10)` to sum the digits.
- **Error 3:** There was a **missing semicolon** after `sum = sum % 10`. Every statement in Java needs to end with a semicolon.
- **Error 4:** The logic to reduce `sum` in the loop was incorrect. The statement should be `sum = sum / 10` to correctly extract the digits.

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

To fix the errors, you would need **4 breakpoints**, one for each error to isolate and debug the issues.

- **Breakpoint 1:** To fix the `while(sum == 0)` condition, set a breakpoint inside the loop to check if the condition behaves as expected.
- **Breakpoint 2:** Set a breakpoint for the incorrect operation `s = s * (sum / 10)` to ensure it's correctly summing digits.
- **Breakpoint 3:** Place a breakpoint after `sum = sum % 10` to verify that the semicolon is added and the syntax is correct.
- **Breakpoint 4:** Set a breakpoint after the division `sum = sum / 10` to verify the correct digit extraction in each loop iteration.

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

- **Step 1:** Corrected the `while(sum == 0)` condition to `while(sum > 0)` to ensure the loop processes digits as expected.
- **Step 2:** Fixed the digit summing operation by changing `s = s * (sum / 10)` to `s = s + (sum % 10)`, correctly adding the digits.
- **Step 3:** Added the missing semicolon after the line `sum = sum % 10` to avoid syntax errors.
- **Step 4:** Corrected the logic for removing the last digit by using `sum = sum / 10` to divide by 10, which extracts digits in a loop.

3. Submit your complete executable code.

```
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;

      // Continue the loop until num reduces to a single-digit number
      while (num > 9)
      {
         sum = num;  // Assign current number to sum for digit extraction
         int s = 0;  // Initialize s to store the sum of digits

         // Extract digits and calculate their sum
         while (sum > 0)
         {
            s = s + (sum % 10); // Add the last digit to s
            sum = sum / 10;     // Remove the last digit
         }

         num = s;  // Assign the digit sum to num for next iteration
      }

      // Check if the final single digit is 1 (Magic number condition)
```

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

        ob.close(); // Close the scanner
    }
}
```

5. Merge Sort

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

There are **5 errors** in the program.

- **Error 1:** In the method `mergeSort`, the line `int[] left = leftHalf(array+1);` is incorrect. The `array+1` operation is not valid in Java. The correct array should be passed as `leftHalf(array)`.
- **Error 2:** Similarly, the line `int[] right = rightHalf(array-1);` is incorrect. The `array-1` is also invalid. It should be `rightHalf(array)`.
- **Error 3:** In the `mergeSort` method, `merge(array, left++, right--);` is incorrect. Incrementing `left++` and decrementing `right--` are inappropriate here. They should just be passed as `merge(array, left, right);`.
- **Error 4:** In the `merge` method, the comment says `Second working version`, which seems like leftover documentation that can cause confusion.

- **Error 5:** The array merging logic works correctly, but it should be clarified that the method works on whole arrays instead of parts, so there is no need for incremental operations like `left++` and `right--`.

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

You need **5 breakpoints** to isolate and debug the errors.

- **Breakpoint 1:** To fix `array+1`, set a breakpoint at the point where `leftHalf(array+1)` is called.
- **Breakpoint 2:** Set a breakpoint at `rightHalf(array-1)` to ensure the correct array is passed.
- **Breakpoint 3:** A breakpoint is required where `merge(array, left++, right--);` is used to ensure no illegal operations are performed on the array.
- **Breakpoint 4:** You can set a breakpoint near the `merge` method to remove unnecessary comments and clean up the logic.
- **Breakpoint 5:** Set a breakpoint during the merge operation to verify that array elements are correctly merged.

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

- **Step 1:** Corrected the array-slicing issues by passing `array` directly to `leftHalf(array)` and `rightHalf(array)`.
- **Step 2:** Fixed the issue of modifying array references (`left++`, `right--`) when passing them to the `merge` function. Arrays should be passed without incrementing/decrementing.
- **Step 3:** Removed the unnecessary comment that confused the `merge` method functionality.

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));
    }

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

            // 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.
    // 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++;
            }
        }
    }
}
```

6. Multiply Matrices

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

The program has **3 errors.**

1. **Array Index Out of Bounds**: The indexing in the nested loop for matrix multiplication is incorrect:
    - The indices should be `first[c][k]` and `second[k][d]` instead of `first[c-1][c-k]` and `second[k-1][k-d]`. This can lead to an `ArrayIndexOutOfBoundsException`.
2. **Improper Calculation Logic**: The multiplication formula should use `sum += first[c][k] * second[k][d];` to accumulate the

product of the corresponding elements instead of the current incorrect form.
3. **Variable Initialization**: The `sum` variable should be reset to 0 at the beginning of the innermost loop to prevent carrying over the sum from previous calculations.

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

To fix the identified errors, you would typically need breakpoints in these areas:

1. **Before the Nested Loop**: To inspect the values of `first`, `second`, and indices before calculating the product.
2. **Inside the Nested Loop**: After the multiplication calculation to check the value of `sum` before storing it in the `multiply` array.
3. **After Matrix Initialization**: To verify that matrices are being populated correctly.

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

Here's how to fix the identified errors:

1. **Correct Indexing**:
    a. Change `first[c-1][c-k]` to `first[c][k]` and `second[k-1][k-d]` to `second[k][d]`.
2. **Reset the Sum**:
    a. Initialize `sum = 0` at the beginning of the loop where the multiplication occurs.
3. **Update Multiplication Logic**:
    a. Modify the multiplication logic to use the correct formula: `sum += first[c][k] * second[k][d];`.

3. Submit your complete executable code.

```
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:");
      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();

         // Matrix multiplication logic
         for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++) {
               sum = 0; // Reset sum for each element of the result matrix
               for (k = 0; k < n; k++) { // Corrected loop condition
                  sum += first[c][k] * second[k][d]; // Correct multiplication logic
               }

               multiply[c][d] = sum;
            }
```

```
        }

        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");
        }
     }

     in.close(); // Close the scanner
  }
}
```

7. Quadratic Probing

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

There are **4 errors** in the program.

1. **Syntax Error in Incrementing `i`**:
   - The expression `i + = (i + h / h--) % maxSize;` contains an incorrect spacing between + and =. It should be `i += (i + h) % maxSize;` and also the calculation of h in the insertion logic should be corrected.
2. **Improper Hash Function**:
   - The hash function might produce negative indices because `key.hashCode()` can return a negative number. You should normalize the result by adding `maxSize` and using modulo.
3. **Infinite Loop Risk**:
   - In the `insert`, `get`, and `remove` functions, the calculation for `i` in the loops could potentially lead to an infinite loop. You should ensure that the increment of h occurs correctly in these methods.
4. **Redundant Code in `remove` Method**:

- After deleting an entry, the rehashing process is inefficient and could lead to errors. It should be revised to ensure proper handling of the probing.

2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

1. **Correct the increment syntax**:
   a. Change `i + = (i + h / h--) % maxSize;` to `i += (h * h++) % maxSize;`.
2. **Normalize the hash value**:
   a. Update the hash function to: `return (key.hashCode() % maxSize + maxSize) % maxSize;` to ensure it's non-negative.
3. **Fix increment logic in loops**:
   a. Ensure the h increment logic is correct in the `insert`, `get`, and `remove` methods.
4. **Streamline the `remove` method**:
   a. Update the rehashing logic after removing a key.

3. Submit your complete executable code.

```
import java.util.Scanner;

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;
}

/** Function to insert key-value pair **/
```

```
public void insert(String key, String val) {
    if (isFull()) {
        System.out.println("Hash table is full. Cannot insert.");
        return;
    }

    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 = (tmp + h * h) % maxSize;
        h++;
    } 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 = (hash(key) + h * h) % maxSize;
        h++;
    }
    return null;
```

```
    }

    /** Function to remove key and its value **/
    public void remove(String key) {
        if (!contains(key))
            return;

        int i = hash(key), h = 1;
        while (!key.equals(keys[i])) {
            i = (i + h * h) % maxSize;
            h++;
        }

        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];
            String 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:
                    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');

        scan.close(); // Close the scanner
    }
}
```

8. Sorting Array

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

There are **5 errors** in the program:

- **Class Name Error**: The class name `Ascending _Order` has a space, which is not valid in Java.
- **Loop Condition Error**: The outer loop uses `i >= n` instead of `i < n`, which causes incorrect loop behavior.
- **Semicolon After Outer Loop**: There is a semicolon after the outer loop definition, which leads to the inner loop not executing correctly.

- **Sorting Logic Flaw**: The condition for swapping elements is incorrect; it uses <= instead of >.
- **Comma Handling in Output**: The print logic does not properly handle commas, leading to an extra comma at the end of the output.

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

- **Breakpoint 1**: Check and correct the class name to remove spaces.
- **Breakpoint 2**: Verify and correct the loop condition in the outer loop.
- **Breakpoint 3**: Remove the unnecessary semicolon after the outer loop.
- **Breakpoint 4**: Correct the sorting logic to ensure proper comparisons for swapping.
- **Breakpoint 5**: Adjust the print logic to handle commas correctly to avoid trailing commas.

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

1. **Fixed Class Name**: Renamed `Ascending _Order` to `AscendingOrder`.
2. **Corrected Loop Condition**: Changed the condition from `i >= n` to `i < n` in the outer loop.
3. **Removed Extra Semicolon**: Deleted the semicolon after the outer loop declaration.
4. **Updated Sorting Logic**: Changed the sorting condition from `if (a[i] <= a[j])` to `if (a[i] > a[j])` to correctly identify when to swap elements.
5. **Refined Print Logic**: Modified the print loop to avoid printing a trailing comma by checking the index against `n - 1`.

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];
        System.out.println("Enter all the elements:");

        // Input elements into the array
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }

        // Sorting the array in ascending order using bubble sort
        for (int i = 0; i < n; i++) {  // Correct loop condition
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {  // Change the condition for sorting
                    // Swap the elements
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }

        // Print the sorted array
        System.out.print("Ascending Order: ");
        for (int i = 0; i < n; i++) {
            System.out.print(a[i]);
            if (i < n - 1) {
                System.out.print(", ");  // Add comma for all but the last element
            }
        }
        System.out.println();  // Print newline at the end
    }
}
```

9. Stack Implementation

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

There are **5 errors** in the program:

1. **Incorrect Index Handling in Push Method**:
    - The `push` method incorrectly decrements `top` before assigning a value to `stack[top]`. It should increment `top` instead.
2. **Incorrect Index Handling in Pop Method**:
    - The `pop` method incorrectly increments `top` when removing an element. It should decrement `top` instead.
3. **Display Method Loop Condition**:
    - The loop in the `display` method uses `i > top`, which is incorrect. It should use `i <= top` to display the elements from the bottom of the stack to the top.
4. **Uninitialized Stack Values**:
    - The stack array is not initialized with default values. It is good practice to handle this in the `display` method, although not strictly an error.
5. **Redundant Stack Size Check**:
    - The stack size is defined in the constructor, but there are no mechanisms (like resizing) to handle situations where more elements are pushed than the size.

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

1. **Breakpoint 1**: Check and correct the index handling in the `push` method.
2. **Breakpoint 2**: Check and correct the index handling in the `pop` method.
3. **Breakpoint 3**: Fix the loop condition in the `display` method.
4. **Breakpoint 4**: Consider implementing stack resizing for better flexibility (optional for basic fixes).

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

1. **Corrected Push Method**: Changed `top--` to `top++` before assigning a value to `stack[top]`.

2. **Corrected Pop Method**: Changed `top++` to `top--` when removing an element.
3. **Fixed Display Method**: Updated the loop condition to `i <= top` to correctly iterate through the stack.
4. **Optional Resizing**: Although resizing is not included in this fix, it's recommended for a robust implementation (not necessary for the question).

3. Complete Executable Code

```
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++; // Increment top
            stack[top] = value; // Assign value to the top of the stack
        }
    }

    public void pop() {
        if (!isEmpty()) {
            top--; // Decrement top
        } else {
            System.out.println("Can't pop...stack is empty");
        }
    }

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

```
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty");
        } else {
            for (int i = 0; i <= top; i++) { // Correct loop 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(); // Display stack elements
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display(); // Display stack elements after popping
    }
}
```

10. Tower Of Hanoi

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

There are **4 errors** in the program:

1. **Incorrect Increment/Decrement in Recursion**:

    - In the second recursive call `doTowers(topN ++, inter--, from+1, to+1)`, using `++` and `--` directly in the function call is incorrect. It should simply be `topN - 1` and `inter` without incrementing or decrementing them in the function call.
2. **Logical Error in the Recursive Calls**:
    - The parameters for the recursive calls are not set correctly. Specifically, when calling `doTowers(topN - 1, from, to, inter);`, the roles of the 'to' and 'inter' pegs should be properly passed in the second recursive call.
3. **Base Case Printing Logic**:
    - The base case should also include the logic for disk 1 moving correctly from the source to the target. The print statement here is fine, but the recursive logic needs proper handling.
4. **Incorrect Output Formatting**:
    - The output of the disk movement does not reflect the increment and proper tracking of which disk is moving when viewed through the recursive calls.

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

1. **Breakpoint 1**: Correct the increment and decrement in the recursive function call.
2. **Breakpoint 2**: Ensure the parameters in the recursive calls match the expected logic for moving disks.
3. **Breakpoint 3**: Adjust the output formatting to clarify the movement of disks correctly.

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

1. **Corrected Increment/Decrement**: Changed `doTowers(topN ++, inter--, from+1, to+1)` to `doTowers(topN - 1, inter, from, to)`.

2. **Adjusted Recursive Logic**: Re-evaluated how parameters are passed during the recursive calls to ensure they correctly reflect the Tower of Hanoi algorithm.
3. **Ensured Clarity of Output**: The output format remains consistent, and the logic flows correctly through recursive calls.

3. Complete Executable Code

```
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3; // Number of disks
        doTowers(nDisks, 'A', 'B', 'C'); // A, B, and C are the names of the rods
    }

    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 {
            // Move topN-1 disks from 'from' to 'inter', using 'to' as the auxiliary
            doTowers(topN - 1, from, to, inter);
            // Move the remaining disk from 'from' to 'to'
            System.out.println("Disk " + topN + " from " + from + " to " + to);
            // Move the disks from 'inter' to 'to', using 'from' as the auxiliary
            doTowers(topN - 1, inter, from, to);
        }
    }
}
```