Jeel Viradiya
202101164
Lab 9 Software Engineering

Question : Tower of Hanoi

Before:

```
//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
```

1. How many mistakes does the program contain? Mention the mistakes you've found:

The program contains a number of errors that can be found:

The line doTowers(topN++, inter--, from+1, to+1) in the doTowers method contains a grammatical mistake. In this case, the increment operators ++ and -- should be avoided.

Instead of using arithmetic operations for the parameters, you should utilize the characters between and from in the same line.

It is necessary to change the line doTowers(topN++, inter--, from+1, to+1) to doTowers(topN - 1, inter, from, to).

2. Which program inspection category do you think is more successful?

"Category E: Control-Flow Errors" and "Category D: Comparison Errors" are very pertinent for locating the errors in this software. This is because the majority of the code's mistakes are associated with recursive mehtod

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

Program inspection is effective at identifying a wide range of errors, but it may not be suitable for finding runtime errors or logic errors that only occur in specific situations. For instance, if the program had incorrect logic that led to the Tower of Hanoi puzzle not being solved correctly, this might not be immediately apparent from code inspection alone.

4. Is the program inspection technique worth applicable?

Yes, program inspection is a valuable technique for identifying and rectifying errors in code. It helps to catch syntax errors, logical errors, and potential issues early in the development process, reducing the likelihood of encountering problems during execution. However, for certain types of errors like runtime errors or performance issues, additional testing and debugging techniques may be necessary.

After: (Complete Executable Code)

```
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3;
        doTowers(nDisks, 'A', 'B', 'C');
    }

    public static void doTowers(int topN, char from, char inter, char to) {
        if (topN == 1) {
            System.out.println("Disk 1 from " + from + " to " + to);
        } else {
            doTowers(topN - 1, from, to, inter);
            System.out.println("Disk " + topN + " from " + from + " to " + to);
            doTowers(topN - 1, inter, from, to);
        }
    }
}
```

Question : Merge Sort

Before:

```
// 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

Answer:

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

Errors in the code:

The code uses incorrect syntax for splitting the array into left and right halves. It uses `array+1` and `array-1`, which is not a valid way to split the array.

The code uses `leftHalf(array+1)` and `rightHalf(array-1)` which would not correctly split the array.

The mergeSort function is not handling the merge correctly.

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

To identify errors in this code, "Category A: Data Reference Errors" would be most effective, as it involves identifying issues related to array manipulation.

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

Program inspection can identify syntax and logic errors, but it may not identify errors that occur at runtime, such as index out-of-bounds errors or null pointer exceptions. These issues often require testing to be discovered.

4. Is the program inspection technique worth applicable?

Program inspection is a valuable technique for identifying issues in code, but it should be complemented with testing and debugging. In this case, program inspection has revealed some syntax and logic errors that need to be corrected.

After: (Complete Executable Code)

java

```
import java.util.Arrays;

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

   public static void mergeSort(int[] array) {
      if (array.length > 1) {
         int size1 = array.length / 2;
         int size2 = array.length - size1;
         int[] left = new int[size1];
         int[] right = new int[size2];

         for (int i = 0; i < size1; i++) {
            left[i] = array[i];
         }
         for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
         }
                  mergeSort(left);
         mergeSort(right);
         merge(array, left, right);
      }
   }
   public static void merge(int[] result, int[] left, int[] right) {
      int i1 = 0; // index into left array
      int i2 = 0; // index into right array

      for (int i = 0; i < result.length; i++) {
         if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
```

```
            result[i] = left[i1]; // take from left
            i1++;
        } else {
            result[i] = right[i2]; // take from right
            i2++;
        }
    }
}
```

Question : Quadratic Probing

Before:

```
/**
 *   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

Answers :

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

Based on the code provided, there don't appear to be any critical errors. However, there may be potential issues not addressed, such as robust input validation, error handling, and comments/documentation that could improve the code's quality.

Category A: Data Reference Errors
- The code seems to correctly handle data reference errors like uninitialized variables and bounds checking.

Category B: Data Declaration Errors
- There are no explicit data declaration errors, such as missing variable declarations.

Category C: Computation Errors
- The code appears to handle arithmetic operations correctly.

Category D: Comparison Errors

- The code seems to correctly handle comparisons between variables.

Category E: Control-Flow Errors
- The code does not appear to contain infinite loops or loops that do not execute.

Category F: Interface Errors
- The code does not show any explicit issues related to function parameters or arguments.

Category G: Input/Output Errors
- The code does not explicitly handle file input/output errors.

Category H: Other Checks
- The code may not be robust in terms of input validation, and it does not explicitly handle potential warnings or informational messages from the compiler.

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

The choice of an effective program inspection category depends on the specific goals of the code review. In this case, ensuring data reference and computation errors, as well as control-flow errors, would be important.

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

The code inspection focuses on certain types of errors, but it does not cover every aspect of program quality. It may not identify performance issues, security vulnerabilities, or issues related to code style and readability.

4. Is the program inspection technique worth applicable?

Program inspection is a valuable technique to improve code quality and identify certain types of errors. It should be used in conjunction with other testing and quality assurance processes to ensure a more comprehensive evaluation of the software.

After: (Complete Executable Code)

```
import java.util.Scanner;

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

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

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

    public int getSize() {
        return currentSize;
    }

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

    public boolean isEmpty() {
        return getSize() == 0;
    }

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

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

public void insert(String key, String val) {
    int tmp = hash(key);
    int i = tmp, h = 1;

    do {
        if (keys[i] == null) {
            keys[i] = key;
            vals[i] = val;
            currentSize++;
            return;
        }
        if (keys[i].equals(key)) {
            vals[i] = val;
            return;
        }
        i = (i + h) % maxSize;
        h++;
    } while (i != tmp);
}

public String get(String key) {
    int i = hash(key);
    int h = 1;
    while (keys[i] != null) {
        if (keys[i].equals(key))
            return vals[i];
        i = (i + h) % maxSize;
        h++;
    }
    return null;
}

public void remove(String key) {
```

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

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

        keys[i] = vals[i] = null;

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

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

public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
```

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

        char ch;
        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. Insert ");
            System.out.println("2. Remove");
            System.out.println("3. Get");
            System.out.println("4. Clear");
            System.out.println("5. Size");

            int choice = scan.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("Enter key and value");
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2:
                    System.out.println("Enter key");
                    qpht.remove(scan.next());
                    break;
                case 3:
                    System.out.println("Enter key");
                    System.out.println("Value = " + qpht.get(scan.next()));
                    break;
                case 4:
                    qpht.makeEmpty();
                    System.out.println("Hash Table Cleared\n");
                    break;
                case 5:
                    System.out.println("Size = " + qpht.getSize());
                    break;
                default:
                    System.out.println("Wrong Entry\n");
                    break;
```

```
            }

            qpht.printHashTable();

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

Question : Multiply Matrices

Before:

```
//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

The code provided has errors related to matrix multiplication logic, but it does not exhibit the errors typically associated with "Category A: Data Reference Errors". The errors in the code are more related to computation and control flow(i.e Category C& Category E).

Certainly, let's review the code for potential errors in other categories as well:

Category B: Data-Declaration Errors
- The code appears to declare and use variables correctly without any obvious declaration errors.

Category C: Computation Errors
- The errors related to matrix multiplication have already been addressed in the previous response.

Category D: Comparison Errors
- There are no comparison errors in the code.

Category E: Control-Flow Errors
- The code does not have control-flow errors such as infinite loops or unreachable code.

Category F: Interface Errors
- The code does not involve interface errors as it doesn't interact with other modules or modules parameters.

Category G: Input / Output Errors
- The code collects input from the user and provides output. However, it's essential to consider error handling for unexpected input or exceptions, but the code doesn't handle such cases.

Category H: Other Checks
- The code lacks robustness checks for input validity, like checking if the provided matrix dimensions are non-negative or ensuring that the matrices are not empty.

1. In the matrix multiplication loop, you are using incorrect indices, resulting in incorrect multiplication. To fix this issue, change `first[c-1][c-k]` to `first[c][k]` and `second[k-1][k-d]` to `second[k][d]`.

2. You should initialize the `sum` variable inside the outer loop to ensure that it starts from zero for each cell in the resulting matrix.

The errors in the code are not related to uninitialized variables, array bounds, or data reference issues. The problems primarily revolve around the computation and logic of matrix multiplication.

After: (Complete Executable Code)

```java
import java.util.Scanner;

class MatrixMultiplication {
   public static void main(String args[]) {
      int m, n, p, q, c, d, k;

      Scanner in = new Scanner(System.in);
      System.out.println("Enter the number of rows and columns of the first matrix");
      m = in.nextInt();
      n = in.nextInt();

      int first[][] = new int[m][n];

      System.out.println("Enter the elements of the first matrix");

      for (c = 0; c < m; c++)
         for (d = 0; d < n; d++)
            first[c][d] = in.nextInt();

      System.out.println("Enter the number of rows and columns of the second matrix");
      p = in.nextInt();
      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 the 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++) {
                int sum = 0; // Initialize sum inside the loop
                for (k = 0; k < p; k++) {
                    sum = sum + first[c][k] * second[k][d];
                }

                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");
        }
    }
  }
}
```

Question : Stack Implementation

Before :

```
//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
```

Answers :

1. How many errors are there in the program? Mention the errors you have identified:
There are three errors in the program: one in Category A (data reference error), one in Category B (data declaration error), and one in Category E (control-flow error).

2. Which category of program inspection would you find more effective?
Program inspections in Categories A and B are still more effective in identifying issues in this code, but Category E (Control-Flow Errors) has also been identified as important.

3. Which type of error you are not able to identify using the program inspection?
Program inspection may not identify all runtime errors that can only be identified during program execution.

4. Is the program inspection technique worth applicable?
Program inspection is a valuable technique for identifying certain types of errors, but it should be combined with actual testing and debugging to ensure program correctness.

After: (Complete Executable Code)

```
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++; // Increment top before pushing the value
            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++) { // Corrected the 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();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

Question : Sorting Arrays

Before :

```
// 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

Answers :

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

There are several errors in the provided program:

a. In the first for loop, the termination condition should be `i < n`, but it is mistakenly written as `i >= n`.

b. There is an extra semicolon after the first for loop, which makes it a standalone statement and causes issues with the sorting logic.

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

The errors in the code can be categorized as follows:

- Category A: Data Reference Errors
  - Error 2: The array subscript is not checked for being within the bounds of the array.
- Category C: Computation Errors

- Error 6: There is a division operation (i/j) without checking if the divisor (j) can be zero.
  - Category E: Control-Flow Errors
    - Error 7: There is an off-by-one error in the first for loop's termination condition (`i >= n` should be `i < n`).
  - Category G: Input/Output Errors
    - No explicit I/O errors, but the program doesn't provide any error handling for invalid inputs or unexpected situations.

3. Which type of error you are not able to identified using the program inspection?
The program inspection technique is effective at identifying errors related to control flow, computation, and basic syntax issues. However, it may not catch more subtle errors related to algorithm correctness, such as whether the sorting algorithm is implemented correctly. It also does not assess the program's logic in terms of producing the expected output.

4. Is the program inspection technique is worth applicable?
Yes, the program inspection technique is worth applying as it helps identify many common programming errors. However, it should be complemented with additional testing and validation methods to ensure the program functions correctly and produces the expected results.

After: (Complete Executable Code)

```
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 the number of elements you want in the 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();
        }

        // Fix the sorting logic
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Changed the comparison to sort in ascending order
                    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.println(a[n - 1]);
    }
}
```

Debugged Parts:
1. The termination condition of the first for loop was fixed: `for (int i = 0; i < n - 1; i++)`
2. The comparison inside the sorting logic was fixed to sort in ascending order: `if (a[i] > a[j])`

Question : Knapsack Problem

Before:

```
//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

This code contains several issues across multiple categories. Let's go through the errors in each category:

Category A: Data Reference Errors

1. The code uses an array `profit` and `weight`, but it doesn't handle cases where the array index is out of bounds.

Category B: Data-Declaration Errors
1. The code uses the `args` array, but it doesn't check if the required command-line arguments are provided. It should verify that `args.length` is at least 2 before attempting to access `args[0]` and `args[1]`.

Category C: Computation Errors
1. There's a logical error in the nested loops when calculating the `option1` and `option2`. The code incorrectly increments the loop variable `n` instead of using the current value of `n`.

Category E: Control-Flow Errors
1. The loops used for calculating `option1` and `option2` have issues. The increment of `n` inside the loops is incorrect and may lead to unexpected behavior.

Category H: Other Checks
1. The code does not handle cases where the command-line arguments are missing or invalid.

Here's the corrected code with these issues addressed:

```java
public class Knapsack {

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: Knapsack <number of items> <maximum weight of knapsack>");
            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 - 1][w];

              // take item n
              int option2 = Integer.MIN_VALUE;
              if (weight[n] <= w)
                 option2 = profit[n] + opt[n - 1][w - weight[n]];

              // select the better of two options
              opt[n][w] = Math.max(option1, option2);
              sol[n][w] = (option2 > option1);
           }
        }

        // determine which items to take
        boolean[] take = new boolean[N + 1];
        for (int n = N, w = W; n > 0; n--) {
           if (sol[n][w]) {
              take[n] = true;
              w = w - weight[n];
           } else {
              take[n] = false;
           }
```

```
        }

        // print results
        System.out.println("Item\tProfit\tWeight\tTake");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
        }
    }
}
```

Question : Magic numbers

Before :

```
// 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.

This code appears to be attempting to check whether a given number is a "magic number." A magic number is defined as a number that, when you repeatedly sum its digits until you reach a single digit, results in 1.

Here are the errors and issues identified in this code:

Category A: Data Reference Errors
1. There is an issue with the loop logic inside the `while` loop. The condition `while (sum == 0)` should be `while (sum != 0)` to calculate the sum of digits correctly. Additionally, there are missing semicolons at the end of the lines within the `while` loop.

Category C: Computation Errors
1. The loop logic within the `while` loop is incorrect. It should sum the digits of `num`, but the code is not correctly performing this operation.

Category D: Comparison Errors
1. There are no comparison errors in the code**.**

Category E: Control-Flow Errors
1. There are no control-flow errors in the code**.**

Category H: Other Checks

1. The code doesn't handle cases where the input number is not a positive integer.

After : Complete Executable Code

```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();

        if (isMagicNumber(n)) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
    }

    // Function to check if a number is a Magic Number
    public static boolean isMagicNumber(int n) {
        while (n > 9) {
            int sum = 0;
            while (n != 0) {
                sum += n % 10;
                n /= 10;
            }
            n = sum;
        }
        return n == 1;
    }
}
```

In this corrected code, the `isMagicNumber` function is added to determine if a number is a magic number, and the issues in the original code are fixed.

Question : GCD AND LCM

Before :

```
//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

1. How many errors are there in the program? Mention the errors you have identified.
The code has errors:
  - In the `gcd` function, the while loop condition should be `while (a % b != 0)` instead of `while (a % b == 0)` to calculate the greatest common divisor (GCD) correctly.
  - In the `lcm` function, the logic is incorrect for calculating the least common multiple (LCM). The LCM calculation should be revised.

2.Which category of program inspection would you find more effective?
The errors in this code can be categorized into Category C: Computation Errors and Category E: Control-Flow Errors.

3. Which type of error you are not able to identify using the program inspection?
While program inspection helps identify various types of errors, it may not be effective in identifying issues related to complex algorithms or mathematical calculations, which could require mathematical validation. In this case, the error related to calculating LCM falls into that category. A deeper understanding of mathematical algorithms is necessary to identify and correct such issues.

4. Is the program inspection technique worth applicable?
Program inspection is a valuable technique for identifying many common programming errors, especially related to syntax, logic, and code structure. However, for complex mathematical algorithms or specialized domain-specific problems like LCM and GCD calculations, additional mathematical validation and testing may be necessary.

After : (Complete Executable Code)

```java
import java.util.Scanner;

public class GCD_LCM {
    static int gcd(int x, int y) {
        while (y != 0) {
            int temp = y;
```

```
            y = x % y;
            x = temp;
        }
        return x;
    }

    static int lcm(int x, int y) {
        return (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();

        int gcdResult = gcd(x, y);
        int lcmResult = lcm(x, y);

        System.out.println("The GCD of two numbers is: " + gcdResult);
        System.out.println("The LCM of two numbers is: " + lcmResult);
        input.close();
    }
}
```

Question : Armstrong

Before :

```
  //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.

Answers :

1. How many errors are there in the program? Mention the errors you have identified.
There is an error in the code:
  - The variable `remainder` is calculated incorrectly, it should be the remainder of `num` divided by 10 instead of `num` divided by 10.
  - The loop should also update the value of `num` with the quotient, not the remainder.
  - The check for Armstrong number is incorrect; it should be `check == n` instead of `check == num`.

2. Which category of program inspection would you find more effective?
The errors identified here fall into different categories. For example, the calculation error with `remainder` falls under "Computation Errors," and the incorrect check for an Armstrong number falls under "Comparison Errors."

3. Which type of error you are not able to identify using the program inspection?
Some semantic errors may not be detected using program inspection alone, such as incorrect logic that leads to incorrect results. Program inspection is primarily useful for identifying syntactical and structural issues.

4. Is the program inspection technique worth applicable?
The program inspection technique is still valuable for catching many types of errors. However, it may not catch all issues, particularly logic errors. For those, testing and debugging are also essential.

After : (Complete Executable Code)

```java
class Armstrong {
    public static void main(String[] args) {
        int num = Integer.parseInt(args[0]);
        int n = num; // To check at the end
        int check = 0, remainder;
        while (num > 0) {
            remainder = num % 10;
            check = check + (int) Math.pow(remainder, 3);
            num = num / 10; // Update num with the quotient
        }
        if (check == n)
```

```
            System.out.println(n + " is an Armstrong Number");
        else
            System.out.println(n + " is not an Armstrong Number");
    }
}
```