LAB 7

202201458

Srushti Makwana

PROGRAM INSPECTION:

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

1. **Division by Zero Error (Category C: Computation Errors):** The program doesn't handle cases where the input list is empty. If someone calls this function with an empty list, it will crash when attempting to divide by zero while calculating the average.
2. **Potential Uninitialized Variable Warning (Category A: Data Reference Errors):** If the input list is empty, the `average` variable may be used without being assigned a value, leading to a runtime error.
3. **Index Error Potential (Category A: Data Reference Errors):** While using `range(len(numbers))` works in the current state, if any modifications are made to the `numbers` list within the loop (which isn't happening now), it could cause an index error. This is more of a caution for future changes.
4. **Type Error with Non-List Inputs (Category A: Data Reference Errors):** The function doesn't validate input types. If a non-list object, like a string or dictionary, is passed in, the function will throw a `TypeError` when trying to use `len(numbers)` or iterate over it.
5. **Floating Point Division Warning (Category C: Computation Errors):** The division operation `total / len(numbers)` could introduce precision issues when working with floating-point numbers. While this isn't strictly an error, it's something to consider in situations where precision is crucial.

6. **Lack of Type Hints or Documentation (Category H: Other Checks):** There are no type hints or docstrings in the function, which makes it harder for others (or even myself later) to understand the expected input and output.

Which category of program inspection would you find more effective?

I believe **Category A: Data Reference Errors** would be the most effective here. It focuses on ensuring variables are properly initialized and checks array lengths and types before any operations are performed. Additionally, **Category H: Other Checks** is also valuable as it encourages awareness of unused variables and compiler warnings, which can improve the robustness of the code.

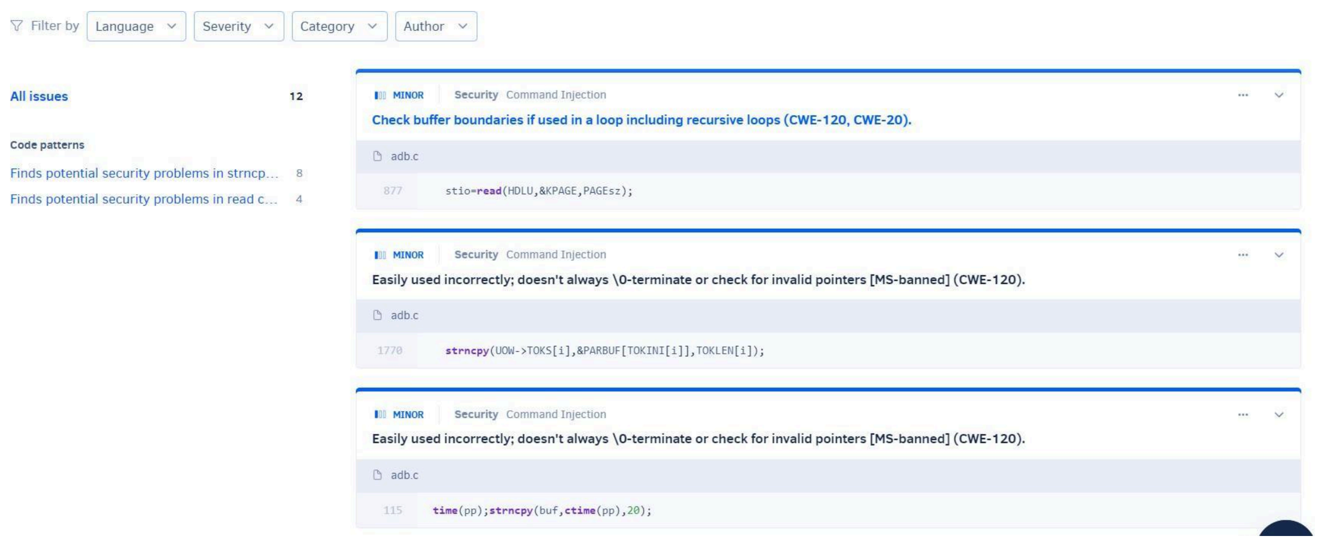
Which type of error are you not able to identify using the program inspection?

One type of error that's difficult to catch with program inspection is **Logic Errors**. These errors don't cause crashes but lead to incorrect results. For example, if non-numeric input is passed, the inspection might not flag it as an error but it could still cause issues later. **Performance Issues** can also go unnoticed, as inspection might not reveal inefficiencies, especially when handling large datasets where more efficient methods could be applied.

Is the program inspection technique worth applying?

Absolutely! Program inspection is definitely worth applying. It helps catch a range of common errors, especially those related to data references, variable declarations, and some logic issues. While it won't catch everything (like logic or performance errors), it's a solid step in the development process that improves software quality and reliability. When combined with thorough testing, this technique can cover all bases and address any gaps.

Static Analysis Tools



Filter by Language Severity Category Author

All issues 12

Code patterns

Finds potential security problems in strncp… 8

Finds potential security problems in read c… 4

MINOR Security Command Injection

Check buffer boundaries if used in a loop including recursive loops (CWE-120, CWE-20).

adb.c

```
stio=read(HDLU,&KPAGE,PAGEsz);
```

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

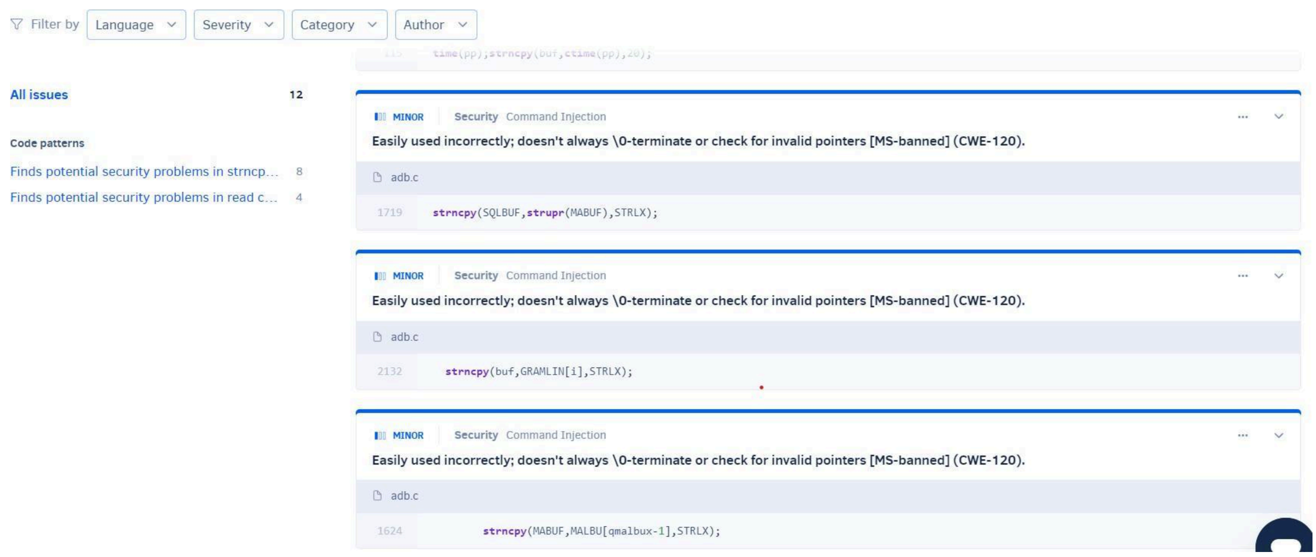
```
strncpy(UOW->TOKS[i],&PARBUF[TOKINI[i]],TOKLEN[i]);
```

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

```
time(pp);strncpy(buf,ctime(pp),20);
```



Filter by Language Severity Category Author

All issues 12

Code patterns

Finds potential security problems in strncp… 8

Finds potential security problems in read c… 4

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

```
strncpy(SQLBUF,strupr(MABUF),STRLX);
```

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

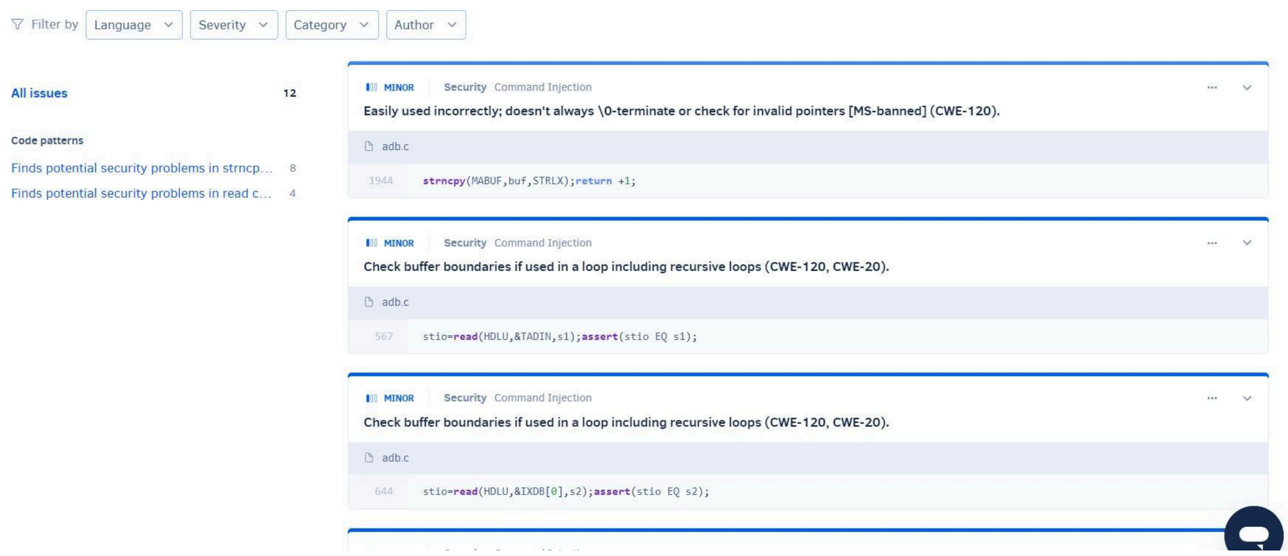
```
strncpy(buf,GRAMLIN[i],STRLX);
```

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

```
strncpy(MABUF,MALBU[qmalbux-1],STRLX);
```



Filter by Language Severity Category Author

All issues 12

Code patterns

Finds potential security problems in strncp… 8

Finds potential security problems in read c… 4

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

```
strncpy(MABUF,buf,STRLX);return +1;
```

MINOR Security Command Injection

Check buffer boundaries if used in a loop including recursive loops (CWE-120, CWE-20).

adb.c

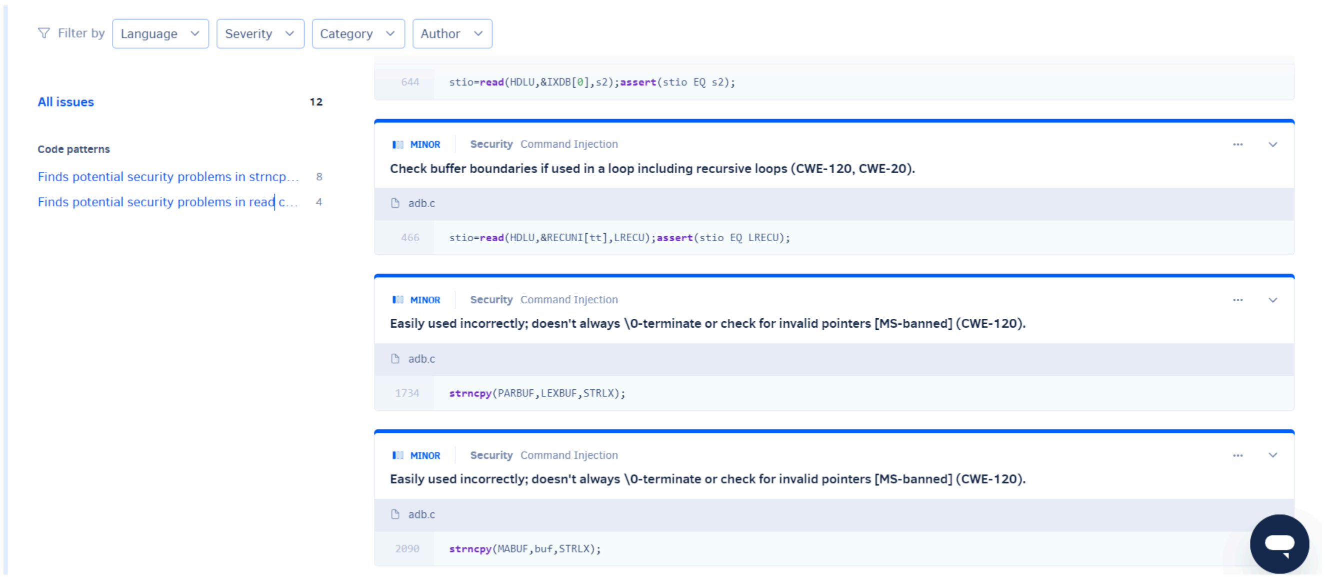
```
stio=read(HDLU,&TADIN,s1);assert(stio EQ s1);
```

MINOR Security Command Injection

Check buffer boundaries if used in a loop including recursive loops (CWE-120, CWE-20).

adb.c

```
stio=read(HDLU,&IXDB[0],s2);assert(stio EQ s2);
```



All issues 12

Code patterns

Finds potential security problems in strncp… 8

Finds potential security problems in read c… 4

```
stio=read(HDLU,&IXDB[0],s2);assert(stio EQ s2);
```

MINOR Security Command Injection

Check buffer boundaries if used in a loop including recursive loops (CWE-120, CWE-20).

adb.c

```
stio=read(HDLU,&RECUNI[tt],LRECU);assert(stio EQ LRECU);
```

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

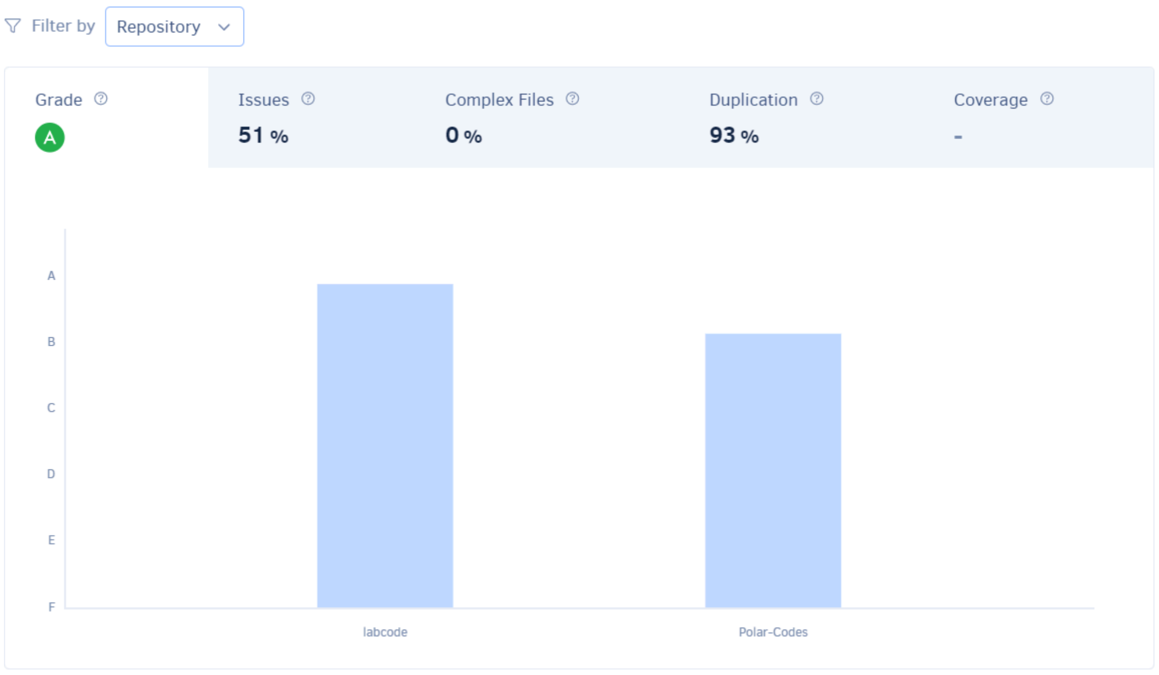
```
strncpy(PARBUF,LEXBUF,STRLX);
```

MINOR Security Command Injection

Easily used incorrectly; doesn't always \0-terminate or check for invalid pointers [MS-banned] (CWE-120).

adb.c

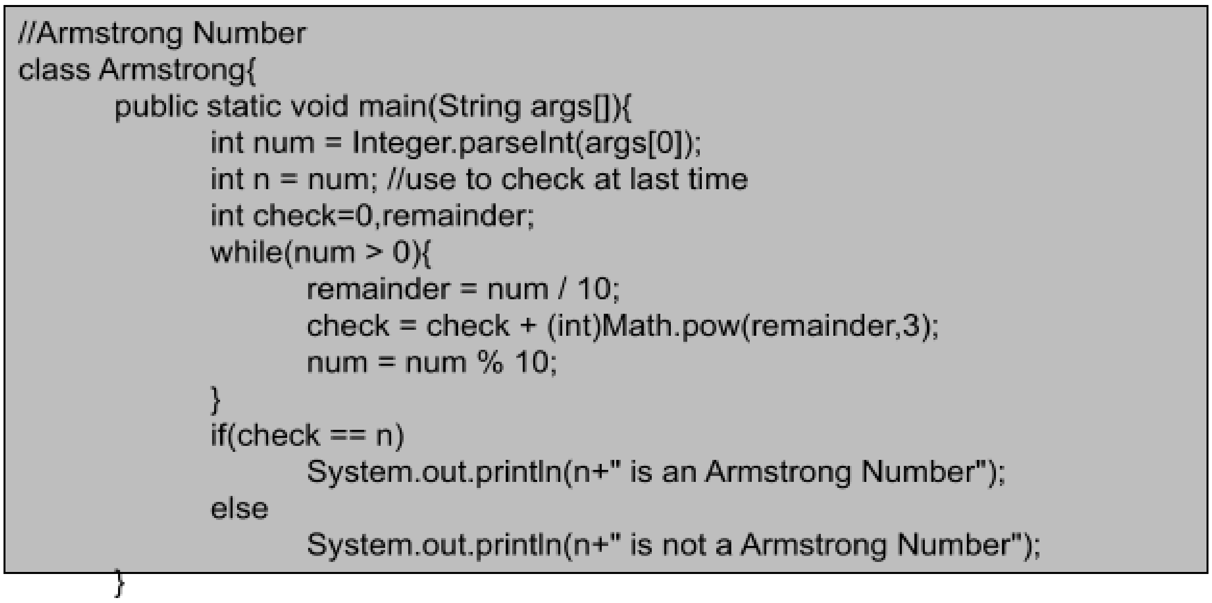
```
strncpy(MABUF,buf,STRLX);
```



Grade	Issues	Complex Files	Duplication	Coverage
A	51 %	0 %	93 %	-

CODE DEBUGGING:

1.



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

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

There are **two main errors** in the code:

1. **Logic Error in Division and Modulus:**
    - The division (remainder = num / 10;) should be finding the remainder (last digit) using the modulus operator %, while the division (num = num % 10;) should be reducing the number by dividing it by 10.
    - Correct usage: remainder = num % 10;

      num = num / 10;

2. **Logic for Armstrong Calculation:**
    - The value of check should be updated with the cube of remainder, but it is mistakenly adding the cube of num instead of remainder.

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

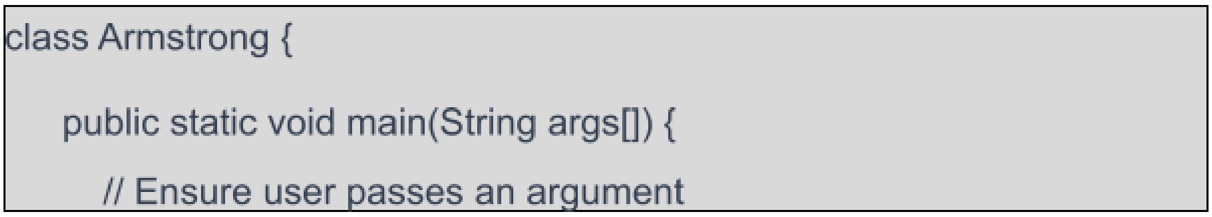
We need **two breakpoints** to investigate and fix the errors:

- **Breakpoint 1:**
  - Before the while loop (while(num > 0){}), we can check the value of num and ensure the calculation logic for remainder and check is correct.
- **Breakpoint 2:**
  - Inside the while loop, after the line remainder = num / 10;, we check if remainder and num are updating as expected.

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

- **Step 1:**
  - Set the breakpoint at the start of the while loop and inspect the value of num and remainder.
  - Realize the modulus (%) should be used instead of division to get the last digit of the number.
- **Step 2:**
  - Update remainder = num % 10; and num = num / 10; to correct the logic for getting the last digit and reducing the number.
- **Step 3:**
  - Debug again to ensure check is correctly calculated as the cube of each digit by using remainder instead of num.

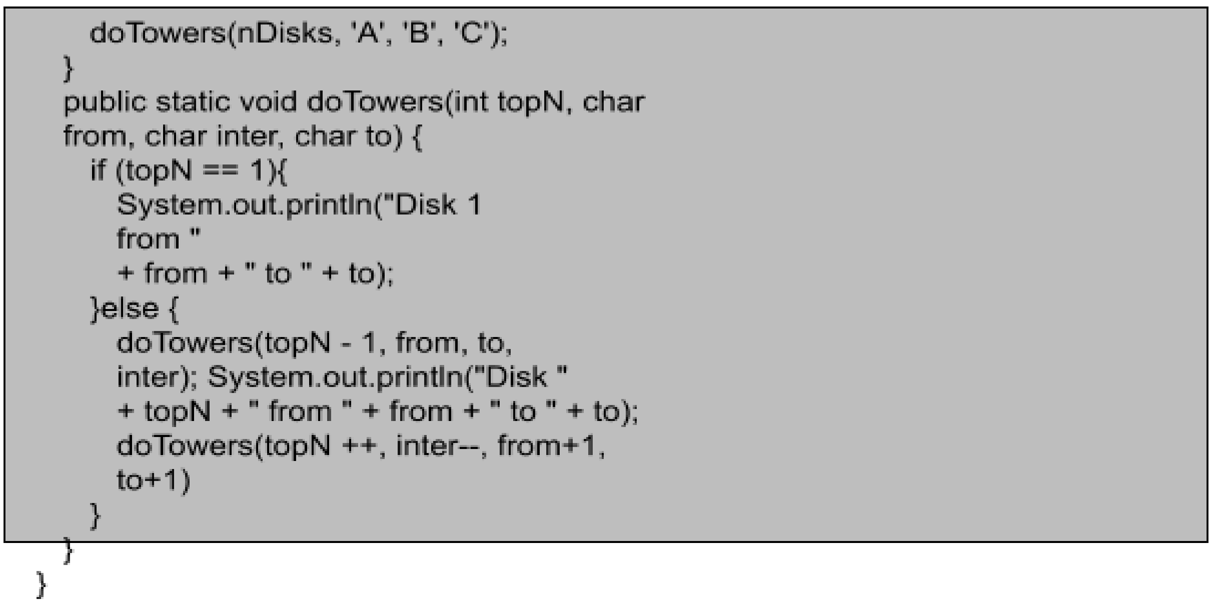
3. Submit your complete executable code:



```
class Armstrong {

    public static void main(String args[]) {
        // Ensure user passes an argument
if (args.length < 1) {

            System.out.println("Please provide a number as
 an argument.");

return;

}

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

int n = num; // Save original number to compare later

int check = 0, remainder;


// Process each digit

        while (num > 0) {

remainder = num % 10; // Get last digit

            check = check + (int)Math.pow(remainder, 3); // Cube the digit
 and add

num = num / 10; // Remove last digit

}

// Check if Armstrong if

        (check == n)

            System.out.println(n + " is an Armstrong

        Number"); else

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

}

}
```

Code 2:

```
//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)
      }
    }
  }
```

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

1. **Infinite Recursion**:
   - In the last line of the doTowers method, you have doTowers(topN ++, inter--, from+1, to+1).
   - topN++ does not correctly reduce the number of disks; it should be topN - 1.
   - inter-- and from+1/to+1 are also incorrect usages, as they are not suitable for characters and will lead to logical errors.
2. **Incorrect Print Statement**:
   - The if (topN == 1) block only prints for disk 1 but does not handle other disks correctly.

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

We need two breakpoints to effectively investigate and fix the errors:

- **Breakpoint 1**:
  - Set a breakpoint at the beginning of the doTowers method. This allows you to check the initial values of topN, from, inter, and to before any operations take place.

- **Breakpoint 2**:
    - Set another breakpoint just before the recursive call for moving the top disks: doTowers(topN - 1, inter, from, to). This will help confirm that the values being passed to the next recursive call are correct.

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

Step 1:

- Set the first breakpoint at the start of the doTowers method to inspect the values of topN, from, inter, and to.
- Identify that topN should decrement for the recursive call to reflect the correct number of disks being moved.

Step 2:

- Correct the recursive call to doTowers(topN - 1, inter, from, to) and remove unnecessary modifications to from and to parameters.

Step 3:

- Confirm the logic for printing the disk movements is correct and that the function handles moving topN-1 disks correctly.

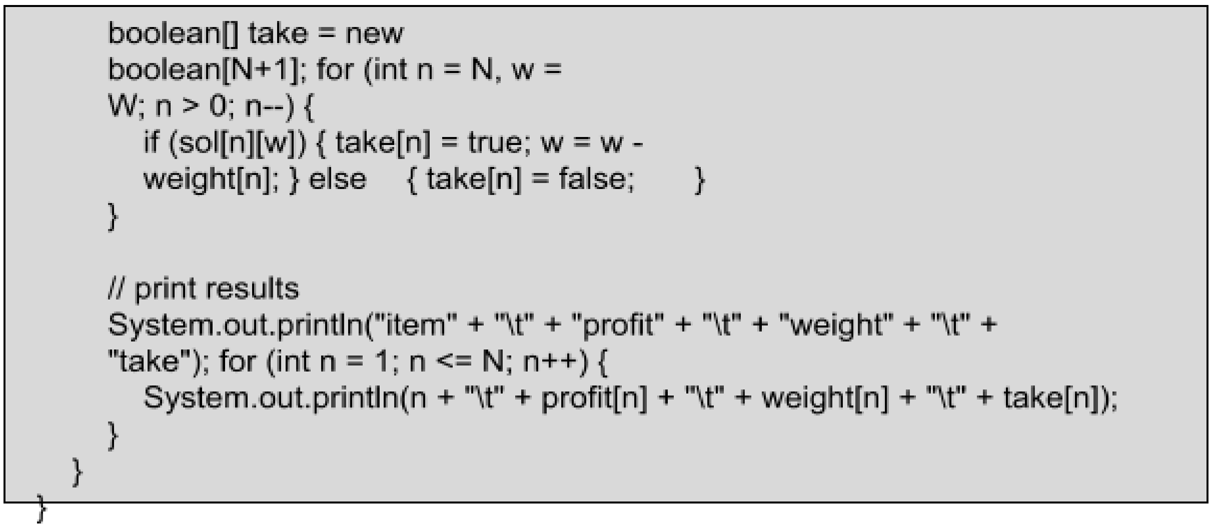
3. Submit your complete executable code:

```
// 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); // Move topN-1 disks from
 'from' to 'inter'
System.out.println("Disk " + topN + " from " + from + " to " + to);
// Move the largest disk to 'to'
```

```
            doTowers(topN - 1, inter, from, to); // Move the topN-1 disks
 from 'inter' to 'to'
}
}
}
```

Code 3:

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

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

There are several errors in the provided Knapsack code:

1. **Incorrect Increment in Loop**:
   - In the inner loop, the statement int option1 = opt[n++][w]; incorrectly increments n, which leads to accessing the wrong index in the opt array.
2. **Incorrect Indexing for Option 2**:
   - The condition if (weight[n] > w) should allow for including the item if it can fit. The profit calculation should also use the correct indexing: it should be profit[n] instead of profit[n-2].
3. **Logic for Determining Which Items to Take**:
   - In the backward traversal to determine which items to take, there's an issue with the logic in updating w. If weight[n] is greater than w, it might skip considering item n even if it was included in the optimal solution.

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

We need three breakpoints to effectively investigate and fix the errors:

- **Breakpoint 1**:

  - Set a breakpoint at the start of the first loop where n iterates from 1 to N. This will help verify that the correct values are being assigned to the opt and sol arrays.

- **Breakpoint 2**:
  - Set another breakpoint inside the inner loop just before calculating option1. This allows you to check the value of n and ensure it's not being incorrectly incremented.

- **Breakpoint 3**:
  - Set a breakpoint in the backward traversal loop to check the values of take and w as items are being selected.

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

Step 1:

- Set the first breakpoint at the start of the first loop and inspect the values of n, profit, and weight.
- Correct the increment issue by changing int option1 = opt[n++][w]; to int option1 = opt[n][w];.

Step 2:

- Fix the logic for calculating option2. Change the line to if (weight[n] <= w) to ensure you only consider the item if it fits in the knapsack, and correct the profit calculation to profit[n] + opt[n-1][w-weight[n]].

Step 3:

- Inspect the backward traversal loop to ensure that w is being updated correctly based on whether the item is included.

3. Submit your complete executable code:

```
public class Knapsack {

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]); // number of items
        int W = Integer.parseInt(args[1]); // maximum weight of knapsack

        int[] profit = new int[N + 1];
        int[] weight = new int[N + 1];
```

```
// Generate random instance, items
1..N for (int n = 1; n <= N; n++) {
    profit[n] = (int) (Math.random() *
    1000); weight[n] = (int)
    (Math.random() * W);
}

// opt[n][w] = max profit of packing items 1..n with weight limit w
// sol[n][w] = does opt solution to pack items 1..n with weight limit w include item n?
int[][] opt = new int[N + 1][W + 1];
boolean[][] sol = new boolean[N + 1][W + 1];

for (int n = 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 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]);
```

Code 4:

```
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.displa
        y();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.displa
        y();
    }
}
```

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

There are several errors in the provided stack implementation:

1. **Incorrect Logic for push Method**:
   - In the push method, top is decremented before assigning the value, which results in an ArrayIndexOutOfBoundsException. The correct logic should increment top after checking if the stack is full.
2. **Incorrect Logic for pop Method**:
   - In the pop method, top is incremented when popping a value, but it should be accessing the value from the stack array before incrementing top. Also, when the stack is empty, it will print an error message but will still increment top in the else block, which is not logical.
3. **Incorrect Loop Condition in display Method**:
   - The loop condition in the display method is incorrect: for(int i=0;i>top;i++). This condition should be i <= top to iterate over the elements correctly.
4. **Display After Pop**:
   - When popping elements, you should display the contents of the stack after some pops, as the display method will show the stack contents after the last operation.

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

You can use the following breakpoints to investigate and fix the errors:

- **Breakpoint 1**:
  - Set a breakpoint in the push method after checking if the stack is full. Inspect the value of top and ensure it updates correctly.
- **Breakpoint 2**:
  - Set a breakpoint in the pop method to check if it correctly accesses and removes the top value from the stack before modifying top.
- **Breakpoint 3**:
  - Set a breakpoint in the display method to verify that the loop condition works properly and all stack elements are displayed.

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

step 1:

In the push method, change top--; to top++; after checking for stack overflow and before assigning the value. Update stack[top] = value;.

Step 2:

- In the pop method, access the stack value before incrementing top. For instance, you can print the popped value.

Step 3:

- Correct the loop condition in the display method from i > top to i <= top.

3. Submit your 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; // top indicates the index of the last element in the stack
    }

    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack is full, can't push a value");
        } else {
            top++; // Increment top to point to the next available index
            stack[top] = value; // Assign value to the stack
        }
    }

    public void pop() {
        if (!isEmpty()) {
            System.out.println("Popped value: " + stack[top]); // Print the
value being popped
            top--; // Decrement top to remove the last element
        } else {
            System.out.println("Can't pop...stack is empty");
        }
    }

    public boolean isEmpty() {
        return top == -1; // Check if the stack is empty
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return;
        }

        System.out.print("Stack elements: ");
        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(5
        0);
        newStack.push(2
        0);
        newStack.push(9
        0);

newStack.display();
        newStack.pop();
        newStack.pop();
newStack.display(); // Display stack after popping
        newStack.pop();
newStack.pop();
newStack.pop(); // This should indicate the stack is empty
        newStack.display(); // Show the empty stack
}
}
```

Code 5:

```
/**

    * 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');

        }

    }
```

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

Syntax Error in Insertion Logic:

- In the insert method, the line i + = (i + h / h--) % maxSize; has a space between + and = which will cause a compilation error. It should be i += (h * h) % maxSize;.

Incorrect Update in Loop:

- In both insert and get methods, i = (i + h * h++) % maxSize; does not correctly implement the quadratic probing. The h should be incremented properly and the calculation of the new index needs to be fixed.

Incorrect Logic in remove Method:

- The line while (!key.equals(keys[i])) might lead to an infinite loop if the key is not found due to incorrect incrementing of i and h.

Clearing the Table:

- In the makeEmpty method, resetting the keys and vals arrays does not actually clear the existing references in the array but just reinitializes them. It should set each position to null.

Redundant Decrement in remove Method:

- There are unnecessary decrements of currentSize in the remove method that should be managed more clearly.

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

You can set the following breakpoints to investigate and fix errors:

- **Breakpoint 1**:
  - Set a breakpoint in the insert method to observe the values of i and h during insertion attempts.
- **Breakpoint 2**:
  - Set a breakpoint in the get method to see how i changes with each probing step.
- **Breakpoint 3**:
  - Set a breakpoint in the remove method to analyze how items are being rehashed and if the currentSize is being decremented correctly.

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

Fix the Syntax:

- Correct i + = to i +=.

Update Probing Logic:

- Change i = (i + h * h++) % maxSize; to i = (i + h * h) % maxSize; h++; to ensure proper incrementing of h after each probing step.

Correct Removal Logic:

- Ensure that remove checks for null entries and doesn't go into an infinite loop if the key does not exist.

Clear the Array Properly:

- In makeEmpty, iterate over the keys and set each index to null.

Review Current Size Management:

- Streamline how currentSize is updated in the remove method.

3. Submit your complete executable code:

```
import java.util.Scanner;

/**
 * Java Program to implement Quadratic Probing Hash Table
 **/
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;
        for (int i = 0; i < maxSize; i++) {
            keys[i] = null;
            vals[i] = null;
        }
    }

    /** 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 Math.abs(key.hashCode()) % maxSize;
}

/** Function to insert key-value pair **/
public void insert(String key, String val) {
    if (isFull()) {
        System.out.println("Hash Table is full, can't 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; // Quadratic probing
        h++;
    } while (i != tmp); // Loop until we come back to the start
}

/** 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; // Quadratic probing
```

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

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

        // Rehash all keys in the same cluster
        for (i = (i + h * h) % maxSize; keys[i] != null; i = (i + h * h) %
maxSize) {
            String tmpKey =
            keys[i]; String tmpVal =
            vals[i]; keys[i] = vals[i] =
            null; currentSize--;
            insert(tmpKey, tmpVal);
        }

        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 **/
```

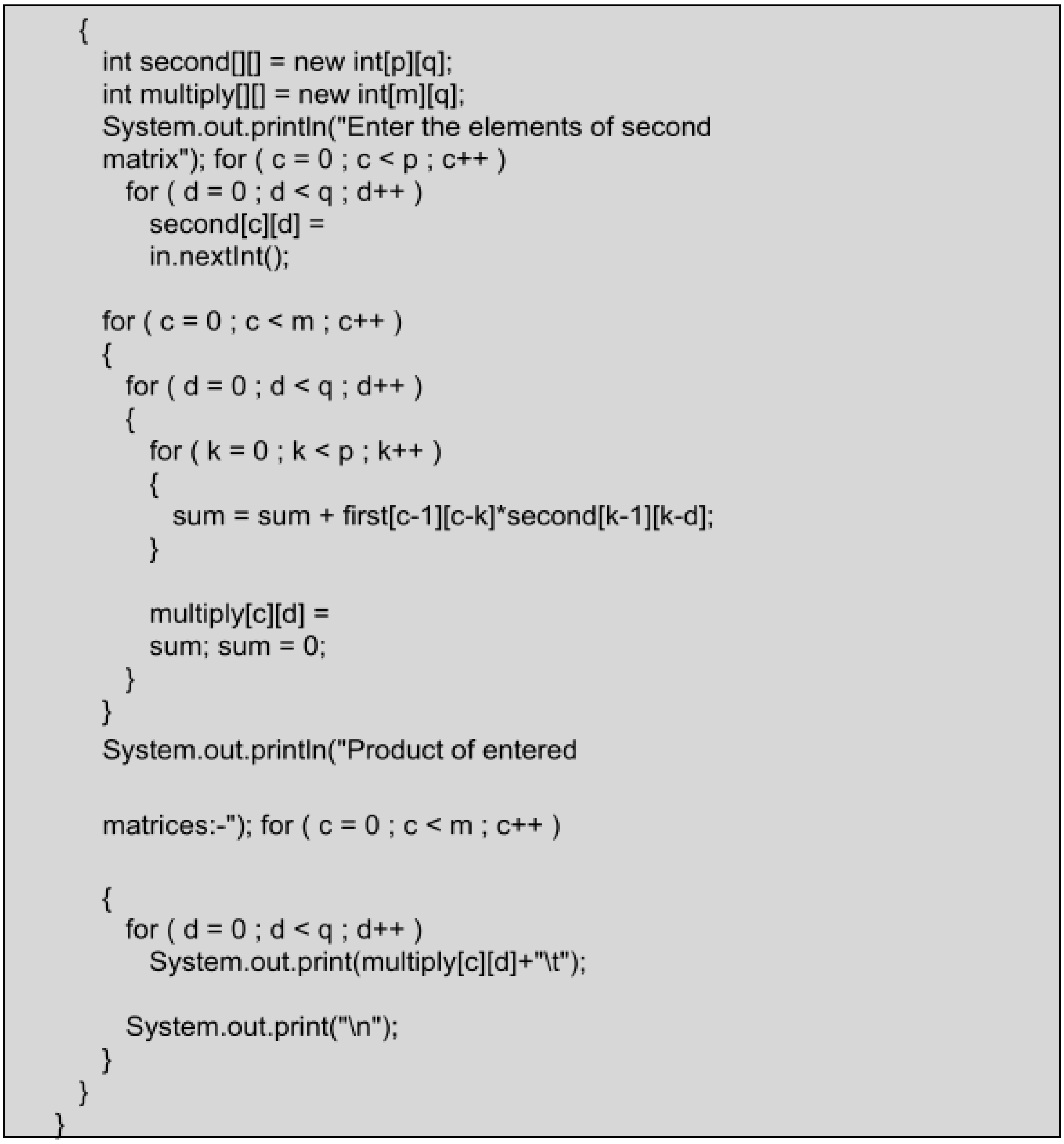
```
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner         scan         =         new
        Scanner(System.in);
        System.out.println("Hash              Table
        Test\n\n");   System.out.print("Enter   size:
        ");
        /**  Create  an  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 to prevent resource leak
}
}
```

Code 6:

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

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

1. **Array Indexing Errors:**
   - In the nested loop for matrix multiplication:

     sum = sum + first[c-1][c-k]*second[k-1][k-d];

     - first[c-1][c-k] and second[k-1][k-d] may cause **ArrayIndexOutOfBoundsException** because array indices start at 0. The correct indices should be first[c][k] and second[k][d].

2. **Logic Error in Indexing:**
   - The loop conditions incorrectly access elements using c-1 and k-1, which do not reflect the correct row and column indices needed for multiplication.
3. **Incorrect Matrix Multiplication Logic:**
   - The inner loop's logic is incorrect because it doesn't correctly accumulate the product of the respective elements.
4. **Uninitialized Variables:**
   - The sum variable is initialized outside the loop and not reset correctly after the multiplication process.

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

- **Breakpoint 1:** Set at the beginning of the outermost multiplication loop to inspect the current values of c, d, and k.
- **Breakpoint 2:** Set inside the innermost multiplication loop just before the line that computes the sum, allowing you to verify the indices and values being accessed in both matrices.

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

step 1:

- Set the first breakpoint at the beginning of the multiplication loops to observe the current values of c, d, and k.

Step 2:

- Change the indexing in the multiplication logic:

  sum = sum + first[c][k] * second[k][d];

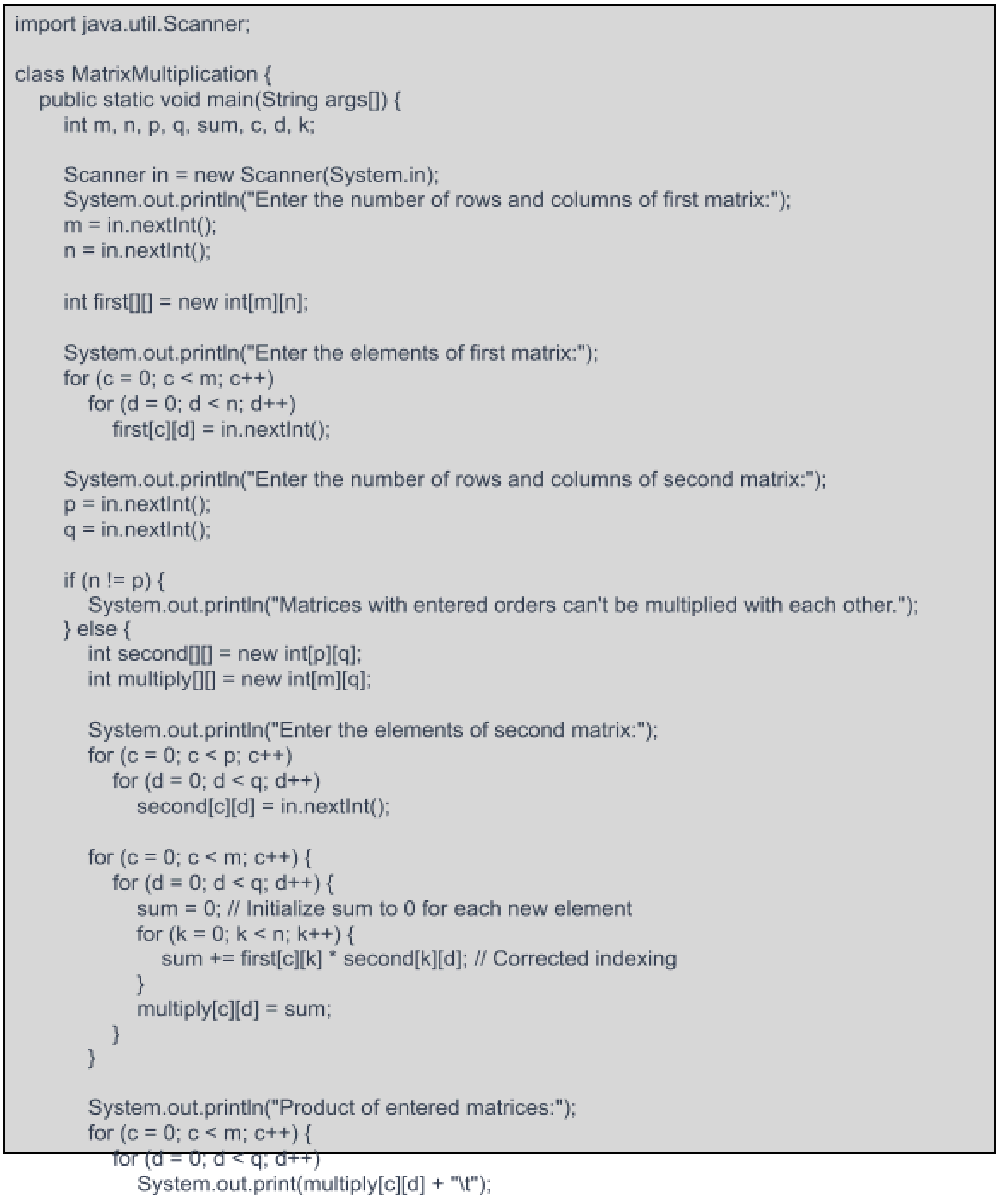
Step 3:

- Ensure that sum is initialized to 0 at the start of each product calculation.

Step 4:

- Check the final multiplication loop to ensure that the results are stored correctly in the multiply array.

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();

         for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++) {
               sum = 0; // Initialize sum to 0 for each new element
               for (k = 0; k < n; k++) {
                  sum += first[c][k] * second[k][d]; // Corrected indexing
               }
               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");
         }
      }
   }
}
```

Code 7:

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

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

Logic Error in Summing Digits:

- The condition in the inner loop is incorrect:

while(sum == 0)

This should be:

while(sum != 0)

- The logic for calculating the sum of the digits is incorrect. The current implementation does not correctly extract and accumulate the digits.

Initialization of Variables:

- The line sum = num; should be placed before the inner loop. It incorrectly initializes sum within the inner loop.

Missing Semicolons:

- There are missing semicolons in the line: sum =

  sum % 10

  It should have a semicolon at the end.

Incorrect Multiplication Logic:

- The logic for extracting digits:

  s = s * (sum / 10);

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

Breakpoint 1: Set at the beginning of the outer loop to check the current value of num before any operations occur.

Breakpoint 2: Set inside the inner loop just before the digit extraction and summation to verify the values being processed.

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

Step 1:

- Set the first breakpoint at the start of the outer loop to inspect the value of num.

Step 2:

- Change the inner loop condition to while(sum != 0) and modify the logic for summing the digits:

Step 3:

- Initialize sum to 0 before entering the inner loop for each iteration.

Step 4:

- Ensure all statements are properly terminated with semicolons and verify the final logic to determine if num equals 1.

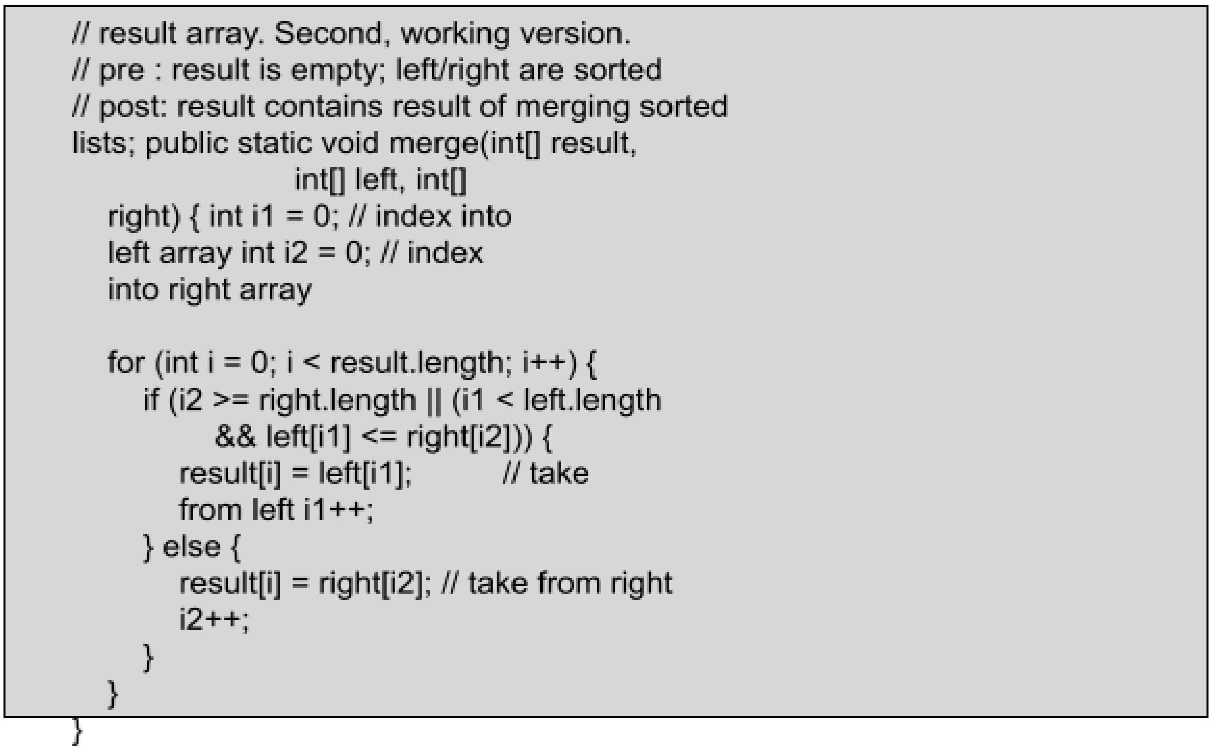
3. Submit your complete executable code:

```
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 num = n;

        while (num > 9) { // Continue until num is a single digit
            int sum = 0; // Initialize sum to 0 for each iteration
            while (num != 0) { // Loop until num becomes 0
                sum += num % 10; // Add the last digit to sum
                num = num / 10; // Remove the last digit
            }
            num = sum; // Update num to the sum of its digits
        }

        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
    }
}
```

Code 8:

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

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

1. **Incorrect Array Slicing:**
   - The methods leftHalf and rightHalf are incorrectly called in mergeSort:

     int[] left = leftHalf(array + 1);

     int[] right = rightHalf(array - 1);

     - This will lead to compilation errors because array + 1 and array - 1 do not correctly slice the array. Instead, it should directly use the array and pass a subarray.
2. **Array Indexing in Merge Method:**
   - The call to merge:

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

     - Using left++ and right-- is incorrect. You should just pass the left and right arrays without modifying them.
3. **Merging Logic in merge:**

    - The merge function should properly handle the length of the result array based on the combined lengths of the left and right arrays.

4. **Handling of Odd-Length Arrays:**
    - The logic for splitting an odd-length array could be improved, as it currently does not correctly handle the additional element.

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

- **Breakpoint 1:** Set at the beginning of the mergeSort method to inspect the current value of the array parameter before any operations take place.
- **Breakpoint 2:** Set inside the merge method to check the indices i1 and i2 and confirm that the merging logic correctly places the elements.

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

Step 1:

- Change the calls to leftHalf and rightHalf to properly slice the array:

  ```
  int[] left = leftHalf(array);
  ```

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

  Step 2:

- In the merge call, pass left and right without using ++ or --.

Step 3:

- Update the logic in merge to ensure it handles the lengths of left and right properly:

  ```
  int[] result = new int[left.length + right.length];
  ```

Step 4:

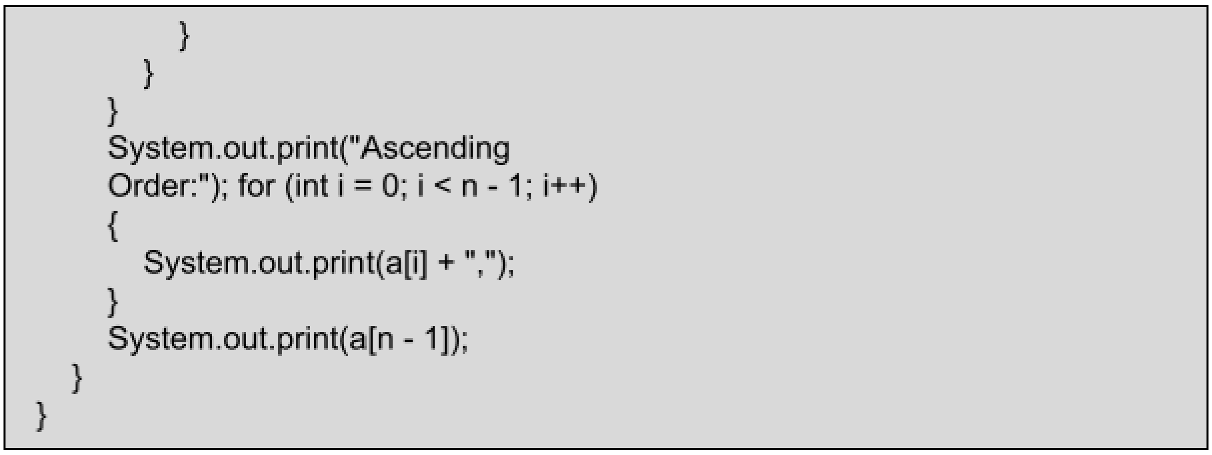
- Adjust leftHalf and rightHalf to ensure they correctly handle odd-length arrays.

3. Submit your complete executable code:

```
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));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    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.
    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0; // index into left array
        int i2 = 0; // index into right array

        // Create a result array with the combined length of left and right
        int[] mergedArray = new int[left.length + right.length];
        int index = 0;

        while (i1 < left.length && i2 <
            right.length) { if (left[i1] <= right[i2]) {
                mergedArray[index++] = left[i1++]; // take from left
            } else {
                mergedArray[index++] = right[i2++]; // take from right
            }
        }

        // Copy any remaining elements from
        left while (i1 < left.length) {
            mergedArray[index++] = left[i1++];
        }

        // Copy any remaining elements from
        right while (i2 < right.length) {
            mergedArray[index++] = right[i2++];
        }

        // Copy merged array back into result
        System.arraycopy(mergedArray, 0, result, 0,
        mergedArray.length);
    }
}
```

Code 9:

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

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

1. **Errors Identified**

   1. **Class Name Error:**
      - The class name Ascending _Order contains a space, which is not allowed in Java class names. It should be a single word, e.g., AscendingOrder.
   2. **Incorrect Loop Condition:**
      - In the outer loop, the condition i >= n should be i < n. This will cause the loop to never execute.
   3. **Extra Semicolon After the Outer Loop:**
      - There is an unnecessary semicolon at the end of the outer loop declaration:

      for (int i = 0; i >= n; i++);

      This terminates the loop prematurely.
   4. **Incorrect Sorting Logic:**
      - The sorting logic is incorrect. The condition if (a[i] <= a[j]) should be if (a[i] > a[j]) for sorting in ascending order.
   5. **Array Printing Logic:**
      - The print statement in the final loop does not correctly handle the formatting. You can simplify this to avoid printing a comma after the last element.

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

- Breakpoint 1: Set at the beginning of the main method to inspect the values of n and the elements of the array before sorting.
- Breakpoint 2: Set inside the sorting loops to monitor the values of a[i] and a[j], ensuring they are being compared and swapped correctly.

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

Step 1:

- Rename the class from Ascending _Order to AscendingOrder.

Step 2:

- Change the outer loop condition from i >= n to i < n.

Step 3:

- Remove the unnecessary semicolon after the outer loop.

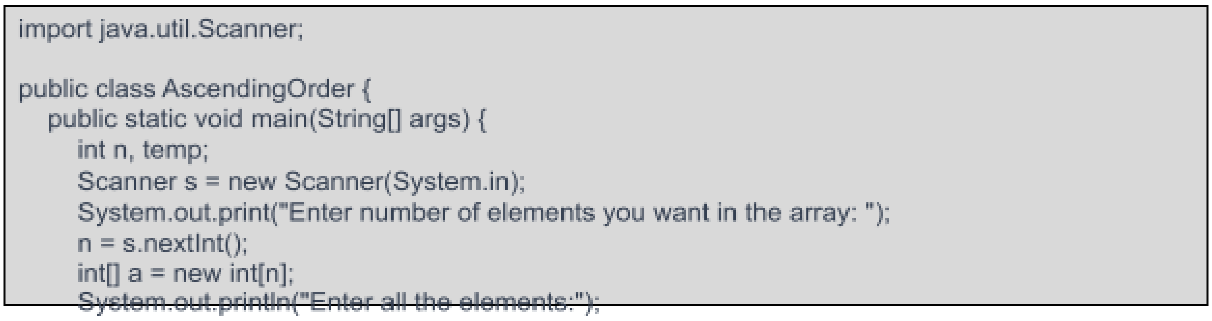
Step 4:

- Update the sorting condition to if (a[i] > a[j]) to ensure correct sorting in ascending order.

Step 5:

- Update the final printing logic to handle commas more elegantly.

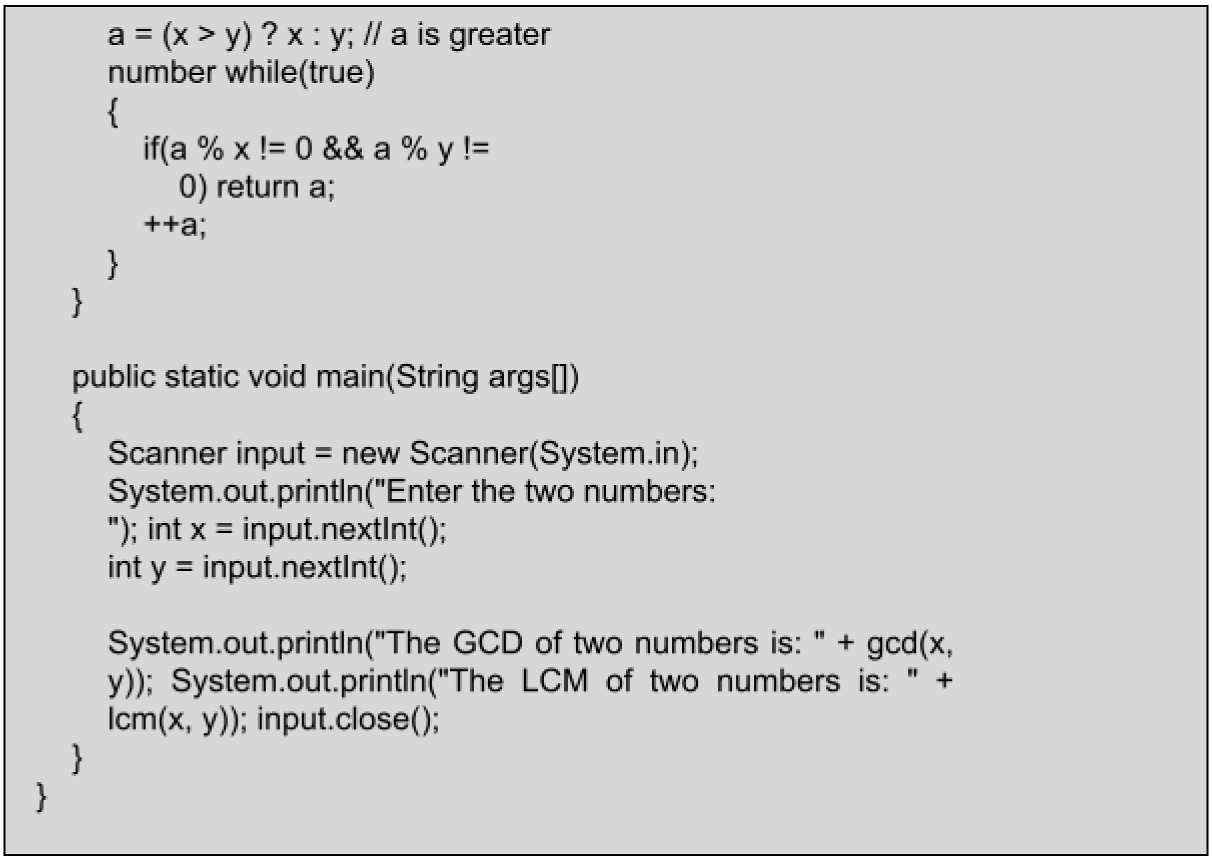
3. **Submit your complete executable code:**



```
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 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();
        }

        // Sort the array in ascending order
        for (int i = 0; i < n; i++) { // Fixed loop
            condition for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Changed to correct sorting condition
                    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(", "); // Print comma for elements except the last
            }
        }
        s.close(); // Close the scanner to prevent resource leaks
    }
}
```

Code 10:

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

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

1. **Errors Identified**

   1. **GCD Calculation Logic:**

      - In the gcd method, the condition in the while loop is incorrect. It should be while (a % b != 0) instead of while (a % b == 0) to correctly implement the Euclidean algorithm.

   2. **LCM Calculation Logic:**

      - The lcm method contains a logical flaw. The condition if(a % x != 0 && a % y != 0) is incorrect. It should be if(a % x == 0 && a % y == 0) to check if a is a multiple of both x and y.

   3. **Inefficient LCM Calculation:**

      - The current implementation of the lcm function is not efficient because it uses a brute force method. It can be optimized by using the formula LCM(x, y) = (x * y) / GCD(x, y).

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

Breakpoint 1: Set at the beginning of the gcd method to inspect the values of x and y.

Breakpoint 2: Set inside the lcm method to verify the values of a and check the condition in the while loop.

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

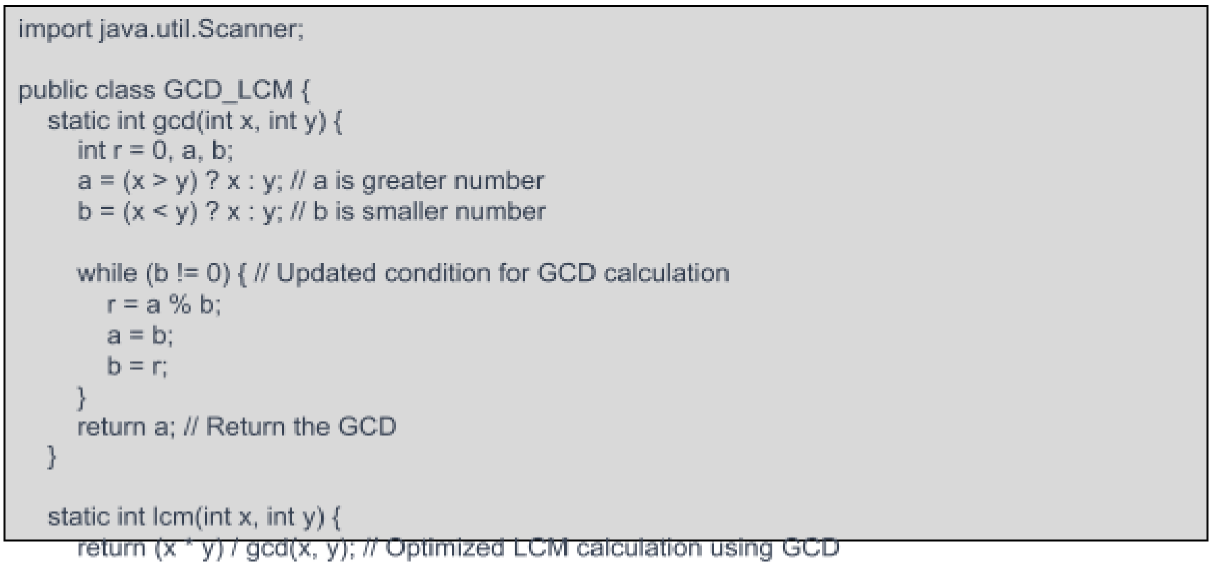
Step 1:

- Correct the condition in the gcd method to while (a % b != 0). Step 2:
- Update the condition in the lcm method to check for multiples correctly: if (a % x == 0 && a % y == 0).

Step 3:

- Optimize the LCM calculation using the GCD function: lcm(x, y) = (x * y) / gcd(x, y).

3. Submit your complete executable code:



```
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

      while (b != 0) { // Updated condition for GCD calculation
         r = a % b;
         a = b;
         b = r;
      }
      return a; // Return the GCD
   }

   static int lcm(int x, int y) {
      return (x * y) / gcd(x, y); // Optimized LCM 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();
    }
}
```