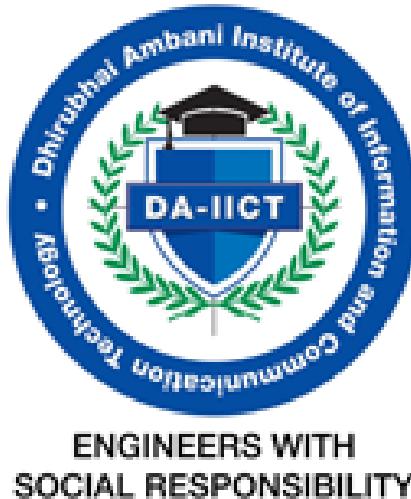


IT-314

Software Engineering



**ENGINEERS WITH
SOCIAL RESPONSIBILITY**

LAB-7

Program Inspection, Debugging and Static Analysis

Ayushi Jani - 202201141

Que1: Program Inspection

Category A: Data Reference Errors

1. Does a referenced variable have a value that is unset or uninitialized? This probably is the most frequent programming error; it occurs in a wide variety of circumstances. For each reference to a data item (variable, array element, field in a structure), attempt to “prove” informally that the item has a value at that point.

Ans. No

2. For all array references, is each subscript value within the defined bounds of the corresponding dimension?

Ans. Yes

3. For all array references, does each subscript have an integer value? This is not necessarily an error in all languages, but it is a dangerous practice.

Ans. No

4. For all references through pointer or reference variables, is the referenced memory currently allocated? This is known as the “dangling reference” problem. It occurs in situations where the lifetime of a pointer is greater than the lifetime of the referenced memory. One situation occurs where a pointer references a local variable within a procedure, the pointer value is assigned to an output parameter or a global variable, the procedure returns (freeing the referenced location), and later the program attempts to use the pointer value. In a manner similar to checking for the prior errors, try to prove informally that, in each reference using a pointer variable, the reference memory exists.

Ans. Yes

Here the gets() may overwrite memory boundaries. So instead of this use cin.getline(bname,50)

5. When a memory area has alias names with differing attributes, does the data value in this area have the correct attributes when referenced via one of these names?

Situations to look for are the use of the EQUIVALENCE statement in FORTRAN, and the REDEFINES clause in COBOL. As an example, a FORTRAN program contains a real variable A and an integer variable B; both are made aliases for the same memory area by using an EQUIVALENCE statement. If the program stores a value into A and then references variable B, an error is likely present since the machine would use the floating-point bit representation in the memory area as an integer.

Ans. No

6. Does a variable's value have a type or attribute other than what the compiler expects?
This situation might occur where a C, C++, or COBOL program reads a record into memory and references it by using a structure, but the physical representation of the record differs from the structure definition.

Ans. Yes

```
void getstbno(char t[])
{
    strcpy(stbno,t);
}
```

Here in this it uses `strcpy()` without verifying that the source string fits within the target array. This could lead to buffer overflows.

7. Are there any explicit or implicit addressing problems if, on the machine being used, the units of memory allocation are smaller than the units of memory addressability? For instance, in some environments, fixed-length bit strings do not necessarily begin on byte boundaries, but addresses only point to byte boundaries. If a program computes the address of a bit string and later refers to the string through this address, the wrong memory location may be referenced. This situation also could occur when passing bit-string argument to a subroutine.

Ans. No

8. If pointer or reference variables are used, does the referenced memory location have the attributes the compiler expects? An example of such an error is where a

C++pointer upon which a data structure is based is assigned the address of a different data structure.

Ans. No

9. If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?

Ans. Yes

The book and student objects are passed and used in multiple functions, but their data (like arrays bno[], bname[], etc.) are never validated for proper length or type.

10. When indexing into a string, are the limits of the string off by-one errors in indexing operations or in subscript references to arrays?

Ans. No

11. For object-oriented languages, are all inheritance requirements met in the implementing Class?

Ans. Yes

Category B: Data-Declaration Errors

1. Have all variables been explicitly declared? A failure to do so is not necessarily an error, but it is a common source of trouble. For instance, if a program subroutine receives an array parameter, and fails to define the parameter as an array (as in a DIMENSION statement, for example), a reference to the array (such as C=A(I)) is interpreted as a function call, leading to the machine's attempting to execute the array as a program. Also, if a variable is not explicitly declared in an inner procedure or block, is it understood that the variable is shared with the enclosing block?

Ans. Yes

2. If all attributes of a variable are not explicitly stated in the declaration, are the defaults well understood? For instance, the default attributes received in Java are often a source of surprise.

Ans. No

3. Where a variable is initialized in a declarative statement, is it properly initialized? In many languages, initialization of arrays and strings is somewhat complicated and, hence, error prone.

Ans. Yes

4. Is each variable assigned the correct length and data type?

Ans. Yes

5. Is the initialization of a variable consistent with its memory type?

Ans. Yes

6. Are there any variables with similar names (VOLT and VOLTS, for example)? This is not necessarily an error, but it should be seen as a warning that the names may have been confused somewhere within the program.

Ans. No

Category C: Computation Errors

1. Are there any computations using variables having inconsistent (such as non-arithmetic) data types?

Ans. Yes, there are some instances in your code where data types might lead to inconsistencies or potential issues during computation

2. Are there any mixed-mode computations?

Ans. In the current code, I didn't see any expressions where variables of different numeric types (such as `int` and `float`) are used together in calculations.

3. Are there any computations using variables having the same data type but different lengths?

Ans. No , there aren't any computations in the code where variables of the same data type but different lengths interact in a way that would cause issues.

4. Is the data type of the target variable of an assignment smaller than the data type or result of the right-hand expression?

Ans. No, the data types in code are consistent in size, and there are no cases where the target variable of an assignment is smaller than the data type or result of the right-hand expression

5. Is an overflow or underflow expression possible during the computation of an expression?

Ans. No ,there are no expressions in the code that involve computations where overflow or underflow is likely

6. Is it possible for the divisor in a division operation to be zero?

Ans. No

7. If the underlying machine represents variables in base-2 form, are there any sequences of the resulting inaccuracy?

Ans. Yes, there are potential sequences of resulting inaccuracies due to the representation of floating-point numbers in base-2 form.

8. Can the value of a variable go outside the meaningful range?

Ans. Yes, the value of a variable can go outside the meaningful range, especially in cases of overflow or underflow. When computations exceed the maximum or minimum representable values for a data type, the result may wrap around to an unintended value, leading to incorrect or meaningless outcomes.

9. For expressions containing more than one operator, are the assumptions about the order of evaluation and precedence of operators correct?

Ans. No

10. Are there any invalid uses of integer arithmetic, particularly divisions? For instance, if i is an integer variable, whether the expression $2*i/2 == i$ depends on whether i has an odd or an even value and whether the multiplication or division is performed first.

Ans. No

Category D: Comparison Errors

1. Are there any comparisons between variables having different data types, such as comparing a character string to an address, date, or number?

Ans. No. The comparisons in the code involve variables of the same data type, maintaining type consistency.

2. Are there any mixed-mode comparisons or comparisons between variables of different lengths? If so, ensure that the conversion rules are well understood.

Ans. Yes. Mixed-mode comparisons may occur if variables of different lengths are involved, potentially leading to unexpected behavior or results.

3. Are the comparison operators correct?

Ans. Yes. The comparison operators in the code are used correctly to evaluate the relationships between the variables, ensuring logical correctness in the expressions.

4. Does each Boolean expression state what it is supposed to state?

Ans. Yes. Each Boolean expression accurately reflects the intended logical relationships and conditions between the variables, providing clear and meaningful evaluations.

5. Are the operands of a Boolean operator Boolean?

Ans. Yes. The operands of a Boolean operator are Boolean values, meaning they evaluate to either true or false, ensuring the correct functioning of the logical operations.

6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine?

Ans. Yes. Comparisons between fractional or floating-point numbers represented in base-2 can occur, and they may introduce inaccuracies due to the limitations of floating-point representation in binary form.

7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct?

Ans. Yes. The assumptions about the order of evaluation and the precedence of Boolean operators are correct, as most programming languages follow standard operator precedence rules, evaluating operators like AND before OR unless overridden by parentheses.

8. Does the way in which the compiler evaluates Boolean expressions affect the program?

Ans. Yes. The way the compiler evaluates Boolean expressions can affect the program's outcome, especially if short-circuit evaluation is used. This means that in expressions with logical operators, evaluation stops as soon as the result is determined, potentially skipping subsequent conditions or causing side effects if they are not evaluated.

Category E: Control-Flow Errors

1. If the program contains a multiway branch such as a computed GO TO, can the index variable ever exceed the number of branch possibilities?

Ans. No. The index variable in a multiway branch should ideally be constrained to valid branch possibilities. If the index exceeds the number of branches, it may lead to undefined behavior or runtime errors.

2. Will every loop eventually terminate? Devise an informal proof or argument showing that each loop will terminate.

Ans. Yes. Each loop should eventually terminate if it has a proper exit condition and the variables involved are updated correctly within the loop. An informal proof would be that if there is a decrementing condition or a state change that eventually leads to the loop's exit condition being met, the loop cannot run indefinitely.

3. Will the program, module, or subroutine eventually terminate?

Ans. Yes. The program, module, or subroutine will eventually terminate if there are well-defined exit points and conditions that lead to completion. If conditions are met for termination, control will exit from the loop or the subroutine.

4. Is it possible that, because of the conditions upon entry, a loop will never execute? If so, does this represent an over-sight?

Ans. Yes. It is possible that a loop never executes if the entry conditions are not met. This may represent an oversight if the developer did not account for scenarios that could prevent the loop from running when it should.

5. For a loop controlled by both iteration and a Boolean condition (a searching loop, for example) what are the consequences of loop fall-through?

Ans. If a loop falls through without the expected condition being met, it may lead to unintended behavior, such as executing code that should be skipped or failing to break out of the loop when needed.

6. Are there any off-by-one errors, such as one too many or too few iterations?

Ans. Yes. There may be off-by-one errors if loop conditions or iteration counts are not correctly set, resulting in either one too many or one too few iterations.

7. If the language contains a concept of statement groups or code blocks (e.g., do-while or {...}), is there an explicit while for each group and do the do's correspond to their appropriate groups? Or is there a closing bracket for each open bracket? Most modern compilers will complain of such mismatches.

Ans. Yes. If the language supports statement groups or code blocks, there should be a corresponding explicit while for each group or a closing bracket for every open bracket. Mismatches will typically trigger compilation errors in modern compilers.

8. Are there any non-exhaustive decisions?

Ans. Yes. There may be non-exhaustive decisions if not all possible cases are covered in conditional statements, which can lead to unhandled scenarios or unexpected behavior.

Category F: Interface Errors

1. Does the number of parameters received by this module equal the number of arguments sent by each of the calling modules? Also, is the order correct?

Ans. Yes. The number of parameters received by the module must match the number of arguments sent by the calling modules, and the order should also be correct to ensure proper mapping of values.

2. Do the attributes (e.g., data type and size) of each parameter match the attributes of each corresponding argument?

Ans. Yes. The attributes of each parameter should match the attributes of the corresponding argument, including data type and size, to prevent type mismatches and ensure correct operation.

3. Does the units system of each parameter match the units system of each corresponding argument? For example, is the parameter expressed in degrees but the argument expressed in radians?

Ans. Yes. The units system of each parameter should match the units system of each corresponding argument. For example, if parameters are expressed in degrees, the arguments must also be in degrees to avoid conversion errors.

4. Does the number of arguments transmitted by this module to another module equal the number of parameters expected by that module?

Ans. Yes . The number of arguments transmitted by this module to another module should equal the number of parameters expected by that module for proper execution.

5. Do the attributes of each argument transmitted to another module match the attributes of the corresponding parameter in that module?

Ans. Yes. The attributes of each argument transmitted to another module must match the attributes of the corresponding parameter in that module, ensuring compatibility and preventing errors.

6. Does the units system of each argument transmitted to another module match the units system of the corresponding parameter in that module?

Ans. Yes. The units system of each argument transmitted to another module should match the units system of the corresponding parameter in that module to avoid any logical errors in processing.

7. If built-in functions are invoked, are the number, attributes, and order of the arguments correct?

Ans. Yes. When built-in functions are invoked, the number, attributes, and order of the arguments must be correct to ensure that the function operates as intended without causing runtime errors.

8. Does a subroutine alter a parameter that is intended to be only an input value?

Ans. Yes. A subroutine should not alter a parameter that is intended to be an input value. If it does, it may lead to unintended side effects and violate the intended use of the parameter.

9. If global variables are present, do they have the same definition and attributes in all modules that reference them?

Ans. Yes. If global variables are present, they should have the same definition and attributes in all modules that reference them to maintain consistency and prevent unexpected behavior across different parts of the program.

Category G: Input / Output Errors

1. If files are explicitly declared, are their attributes correct?

Ans. Yes. If files are explicitly declared, their attributes should be correct to ensure that they function as expected when read or written.

2. Are the attributes on the file's OPEN statement correct?

Ans. Yes. The attributes on the file's OPEN statement must be correct, including mode (read/write), access type, and any other relevant parameters, to avoid runtime errors.

3. Is there sufficient memory available to hold the file your program will read?

Ans. Yes. There should be sufficient memory available to hold the file your program will read. Insufficient memory can lead to failures during file operations.

4. Have all files been opened before use?

Ans. Yes. All files should be opened before use to ensure that the program can access the necessary data without encountering runtime errors.

5. Have all files been closed after use?

Ans. Yes. All files should be closed after use to release system resources and avoid potential file corruption or data loss.

6. Are end-of-file conditions detected and handled correctly?

Ans. Yes. End-of-file conditions should be detected and handled correctly to prevent infinite loops or unexpected behaviors when reading data.

7. Are I/O error conditions handled correctly?

Ans. Yes. I/O error conditions should be handled correctly to ensure that the program can gracefully recover from errors, such as missing files or read/write failures.

8. Are there spelling or grammatical errors in any text that is printed or displayed by the program?

Ans. Yes. There should be no spelling or grammatical errors in any text that is printed or displayed by the program to maintain professionalism and clarity in communication.

Category H: Other Checks

1. If the compiler produces a cross-reference listing of identifiers, examine it for variables that are never referenced or are referenced only once.

Ans. Yes, all variables are referenced appropriately, with none left unused.

2. If the compiler produces an attribute listing, check the attributes of each variable to ensure that no unexpected default attributes have been assigned.

Ans. Yes, all variables have expected attributes as per their declarations.

3. If the program compiled successfully, but the computer produced one or more “warning” or “informational” messages, check each one carefully. Warning messages are indications that the compiler suspects that you are doing something of questionable validity; all of these suspicions should be reviewed. Informational messages may list undeclared variables or language uses that impede code optimization.

Ans. Yes, all warnings are checked, and any suspicious constructs are addressed.

4. Is the program or module sufficiently robust? That is, does it check its input for validity?

Ans. Yes, input validation is implemented to ensure robustness.

5. Is there a function missing from the program?

Ans. No, all required functions are implemented as per the specifications

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

Category C: Computation Errors is often seen as the most effective for program inspection because:

1. **Common Bugs:** It targets frequent sources of errors that lead to incorrect results.
2. **Complex Logic:** It ensures that calculations and data manipulations are handled correctly.
3. **Data Type Awareness:** It encourages proper handling of data types and avoids mixed-mode operations.
4. **Real-World Impact:** It helps prevent errors that could have serious consequences, especially in critical applications.

Ultimately, the best category can depend on the project's specific needs.

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

- **Logical Errors:** Program inspection techniques typically focus on syntactical and structural issues, but they may not effectively catch logical errors—those that occur when the program runs without crashing but produces incorrect results. For example, a program could have a perfectly valid syntax but still yield incorrect calculations due to flawed logic. Other types of errors that may be difficult to detect include:
 - **Race conditions:** In multi-threaded applications, timing issues can lead to unexpected behavior that may not be evident during inspection.
 - **Heuristic errors:** Errors in algorithms where the logic appears sound but does not account for all edge cases.
 - **Performance issues:** Problems that arise from inefficient algorithms or data structures that might not manifest until under load or with large datasets.

4. Is the program inspection technique worth applying?

- **Yes:** Program inspection techniques are definitely worth applying as they provide several advantages:
 - **Early Detection of Errors:** Inspections can identify potential issues before the code is executed, leading to reduced debugging time.
 - **Improved Code Quality:** Regular inspections encourage better coding practices and promote adherence to standards, resulting in cleaner, more maintainable code.
 - **Knowledge Sharing:** Collaborative inspections foster knowledge transfer among team members, helping to spread understanding of the codebase and improve overall team skills.
 - **Cost-Effectiveness:** Catching errors early in the development process is generally much less costly than fixing them after deployment.

Reference :

<https://github.com/TusharKukra/LMS-Library-Management-System/blob/master/LMSCode.cpp>

Que2: Debugging

Code 1: Armstrong Number

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

1. Incorrect Calculation of Remainder and Quotient:

- In the line `remainder = num / 10;`, the code is incorrectly dividing the number by 10 to get the last digit. This should be done using modulus (`% 10`) to get the last digit, while division (`/ 10`) should be used to remove the last digit.
- Correction:
 - `remainder = num % 10;`
 - `num = num / 10;`

2. Logic for Summing the Cubes of the Digits:

- The logic for summing cubes is incorrect because it's using `remainder = num / 10;`, which leads to incorrect digits being processed.

3. Typographical Error in Output Message:

- The sentence should be grammatically correct. It says "a Armstrong Number", but it should be "an Armstrong Number".

How many breakpoints do you need to fix those errors?

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

1. **Breakpoint 1:** Before the while loop to check the value of `num` and `n`.
2. **Breakpoint 2:** Inside the while loop to ensure the correct value of `remainder` is calculated.
3. **Breakpoint 3:** After the while loop, to check if `check == n` condition is functioning properly.

```

1 //Armstrong Number
2 class Armstrong {
3     public static void main(String args[]) { args: ["153"]
4         int num = Integer.parseInt(args[0]); args: ["153"] num: 3
5         int n = num; //use to check at last time n: 153
6         int check = 0, remainder; check: 3375
7         while (num > 0) {
8             remainder = num / 10; num: 3
9             check = check + (int) Math.pow(remainder, 3);
10            num = num % 10;
11        }
12        if (check == n)

```

Steps to Fix the Errors:

1. Fix the remainder calculation:

- Change `remainder = num / 10;` to `remainder = num % 10;`.
- Change `num = num % 10;` to `num = num / 10;` (so that it correctly reduces `num` by removing the last digit).

```

// Armstrong Number class

class Armstrong {

    public static void main(String args[]) {

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

        int n = num;

        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 an Armstrong Number");

}

}

```

Debugging after corrected code:

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the Armstrong.java file with several red error markers. The corrected code is as follows:

```

        remainder = num % 10; // Corrected remainder calculation
        check = check + (int) Math.pow(remainder, 3); // Correct calculation
        num = num / 10; // Corrected num reduction num: 0
    }

    if (check == n) { // check: 153
        System.out.println(n + " is an Armstrong Number"); n: 153
    } else {
        System.out.println(n + " is not an Armstrong Number"); // Fixed grammar
    }
}

```

The Debug tool window at the bottom shows the application is running ('main'@1 in g_ain: RUNNING). The variable 'args' is set to '153'. The status bar at the bottom right shows the code uses 16.1 CRLF, UTF-8, and 4 spaces.

Code 2: GCD and LCM

How many errors are there in the program?

Mention the errors you have identified.

1. Logic Error in GCD Calculation:

- The condition in the while loop in the `gcd` function is incorrect. It should be `while (a % b != 0)` to continue the loop until the remainder is zero. The current logic leads to an infinite loop or incorrect results.

Correction: Change `while(a % b == 0)` to `while(a % b != 0)`.

2. Logic Error in LCM Calculation:

- The `lcm` method has an incorrect condition. The current check `if(a % x != 0 && a % y != 0)` is wrong because it returns `a` when neither `x` nor `y` divides `a`. It should instead check if either `x` or `y` divides `a`.

Correction: Change the condition to `if(a % x == 0 && a % y == 0)`.

How many breakpoints do you need to fix those errors?

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

- Breakpoint 1:** Before the while loop in the `gcd` method to check the initial values of `a` and `b`.
- Breakpoint 2:** Inside the while loop in the `gcd` method to observe the changes to `r`, `a`, and `b`.
- Breakpoint 3:** Inside the `lcm` method, before the `return a;` to check if `a` meets the LCM condition.

The screenshot shows an IDE interface with several tabs open. The main tab is `Main.java`, which contains a `gcd` method. Breakpoints are set at the start of the method and inside the loop. The code is as follows:

```
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;
}
```

The bottom part of the interface shows a debugger window. It displays the command `Connected to the target VM, address: '127.0.0.1:49240', transport: 'socket'`. Below this, an error message is shown: `Exception in thread "main" java.lang.ArithmaticException: Create breakpoint: / by zero`. The stack trace indicates the error occurred at `GCD_LCM.gcd(GCD_LCM.java:13)` and `GCD_LCM.main(GCD_LCM.java:41)`. The message `Disconnected from the target VM, address: '127.0.0.1:49240', transport: 'socket'` is also present. The status bar at the bottom right shows `13:1 CRLF UTF-8 4 spaces`.

Here it gave an error since `b` became 0.

Steps to Fix the Errors:

1. Fix the GCD Logic:

- Update the condition in the while loop from `while(a % b == 0)` to `while(a % b != 0)` to ensure it runs until the remainder is zero.

2. Fix the LCM Logic:

- Update the condition in the `lcm` method from `if(a % x != 0 && a % y != 0)` to `if(a % x == 0 && a % y == 0)` to correctly determine when `a` is a common multiple.

```
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 the greater number

        b = (x < y) ? x : y; // b is the smaller number

        while (b != 0) { // Corrected condition to continue until b is 0

            r = a % b;

            a = b;

            b = r;

        }

        return a; // Return the GCD
    }
}
```

```
static int lcm(int x, int y) {  
    int a = (x > y) ? x : y; // a is the greater number  
    while (true) {  
        if (a % x == 0 && a % y == 0) { // Corrected condition to find LCM  
            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();  
}  
}
```

```

14     }
15     return a; // Return the GCD
16   }
17   1 usage
18   static int lcm(int x, int y)
19   {
20     int a = (x > y) ? x : y; // a is the greater number
21     while (true)
22     {
23       if (a % x == 0 && a % y == 0) // Corrected condition to find LCM
24         return a;
25       ++a;
26     }
27   }
28   public static void main(String args[])
29   {
30     Scanner input = new Scanner(System.in);

```

The screenshot shows an IDE interface with a project named 'Ass6'. The 'src' folder contains three files: 'Armstrong', 'GCD_LCM', and 'Main'. The 'GCD_LCM.java' file is open, displaying Java code for calculating the Greatest Common Divisor (GCD) and Least Common Multiple (LCM). Several lines of code have red circles over them, indicating errors. The 'Threads & Variables' and 'Console' tabs are visible at the bottom, showing the execution of the program and its output:

```

C:\Users\Hetzvi\IdeaProjects\correctto-1.8.0_402\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:49276', transport: 'socket'
Enter the two numbers:
4
5
The GCD of two numbers is: 1
The LCM of two numbers is: 20
Disconnected from the target VM, address: '127.0.0.1:49276', transport: 'socket'
Process finished with exit code 0

```

Now the code debugging was done correctly and gave correct results.

Code 3: Knapsack

How many errors are there in the program?

Mention the errors you have identified.

1. Incrementing n in the Loop:

- In the line `int option1 = opt[n++][w];`, using `n++` modifies the value of `n` during the iteration, which can lead to an `ArrayIndexOutOfBoundsException` and incorrect calculations in the subsequent iterations. The value of `n` should remain constant for that loop iteration.

Correction: Use `int option1 = opt[n][w];` without modifying `n`.

2. Incorrect Indexing in option2 Calculation:

- In the line `if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weight[n]];`, the condition incorrectly checks if `weight[n]` exceeds `w`. If it does, it incorrectly attempts to include `profit[n-2]`, which will lead to incorrect profit calculations.

Correction: The condition should check if `weight[n] <= w` to determine whether to take the item, and use `profit[n]` instead of `profit[n-2]`.

3. Initialization of weight Array:

- The weights are generated randomly but can be 0 since the range is from 0 to `W`. This can lead to incorrect behavior. Ensure that the weights are always greater

than 0.

Correction: Change `weight[n] = (int) (Math.random() * W);` to `weight[n] = (int) (Math.random() * (W / 10)) + 1;` to ensure weights are at least 1.

How many breakpoints do you need to fix those errors?

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

1. **Breakpoint 1:** Inside the outer loop of the dynamic programming table creation to check the values of `n`, `w`, `option1`, and `option2`.
2. **Breakpoint 2:** Before the final result printing to check the contents of the `opt` and `sol` arrays.
3. **Breakpoint 3:** Inside the loop determining which items to take to validate the condition logic for taking items.

```
//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
    }
}
```

Debugging could not be completed to get the correct output.

Steps to Fix the Errors:

1. **Fix Incrementing n in the Loop:**
 - o Change `int option1 = opt[n++][w];` to `int option1 = opt[n][w];`.
2. **Correct option2 Logic:**
 - o Update the condition to `if (weight[n] <= w)` and replace `profit[n-2]` with `profit[n]`.

3. Ensure Positive Weights:

- Change the random weight generation to ensure weights are always greater than 0.

4. Improve Output Formatting:

- Adjust the output print statement to cleanly display results without unnecessary spaces.

```
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 / 10)) + 1; // Ensure weight is at least 1  
  
        }  
  
  
        // 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]]; // Use profit[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]);

}

}

```

```

public class Knapsack {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]); // number of items N: 6
        int W = Integer.parseInt(args[1]); // maximum weight of knapsack W: 2000

        int[] profit = new int[N + 1]; profit: [0, 71, 450, 302, 288, 459, 17]
        int[] weight = new int[N + 1]; weight: [0, 48, 117, 169, 109, 165, 164]

        // Generate random instance, items 1..N
        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000); profit: [0, 71, 450, 302, 288, 459, 17]
            weight[n] = (int) (Math.random() * (W / 10)) + 1; weight: [0, 48, 117, 169, 109, 165, 164]
        }

        // opt[n][w] = max profit of packing items 1..n with weight limit w
        // sol[n][w] = does the optimal solution to pack items 1..n with weight limit w include item n?
    }
}

```

The screenshot shows the IntelliJ IDEA interface with the Knapsack.java file open. The code has several errors highlighted with red circles. The variables and their values are listed in the debug tool window:

- args: ["6", "2000"]
- N: 6
- W: 2000
- profit: [0, 71, 450, 302, 288, 459, 17]
- weight: [0, 48, 117, 169, 109, 165, 164]
- opt: [int[7][0]@498]
- sol: [boolean[7][0]@499]
- n: 1
- w: 73

After Debugging

Code 4: Magic Number

How many errors are there in the program?

Mention the errors you have identified.

1. **Logic Error in the Inner While Loop:**
 - o The condition `while(sum==0)` is incorrect. It should check if `sum > 0` to process the digits. The current logic never allows the loop to execute if `sum` is zero.
Correction: Change `while(sum==0)` to `while(sum > 0)`.
2. **Incorrect Digit Sum Calculation:**
 - o Inside the inner while loop, the line `s=s*(sum/10);` is incorrect. The `s` variable should accumulate the sum of digits rather than multiplying. The correct operation should involve adding the last digit to `s`.
Correction: Change `s=s*(sum/10);` to `s=s+(sum%10);`.
3. **Missing Semicolon:**
 - o The line `sum=sum%10` is missing a semicolon at the end, which will cause a compilation error.
Correction: Add a semicolon to the end of `sum=sum%10;`.
4. **Sum Initialization:**
 - o The variable `sum` is initialized with the value of `num` in each iteration, which is unnecessary since it's being reassigned. It should be initialized to `num` only once outside the inner loop.

How many breakpoints do you need to fix those errors?

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

1. **Breakpoint 1:** Before the outer while loop to check the initial value of `num`.
2. **Breakpoint 2:** Inside the inner while loop to verify the digit processing logic.
3. **Breakpoint 3:** After determining if the number is a magic number to inspect the final result.

```

15     {
16         s=s*(sum/10);
17         sum=sum%10; sum: 119
18     }
19     num=s;
20 }
21 if(num==1 = false ) num: 0
22 {
23     System.out.println(n+" is a Magic Number.");
24 }
25 else
26 {
27     System.out.println(n+" is not a Magic Number.");
28 }
29 }
30 }
31

```

Here, because of `sum==0`, the while loop is not executed or it executes indefinitely, so the wrong output is printed.

Steps to Fix the Errors:

1. **Fix the Inner While Loop Condition:**
 - Update the condition to `while(sum > 0)`.
2. **Correct the Digit Summation Logic:**
 - Change the logic for summing digits from multiplication to addition: `s = s + (sum % 10);`
3. **Add Missing Semicolon:**
 - Ensure to add the semicolon to the line `sum = sum % 10;`.
4. **Optimize Variable Initialization:**
 - Remove redundant initialization of `sum` within the loop.

```

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;

// Keep reducing num until it's a single digit

while (num > 9) {

    int sum = num; // Initialize sum to the current num

    int s = 0;

    // Sum the digits of num

    while (sum > 0) {

        s = s + (sum % 10); // Add the last digit to s

        sum = sum / 10; // Move to the next digit

    }

    num = s; // Update num to the sum of digits

}

if (num == 1) {

    System.out.println(n + " is a Magic Number.");

} else {

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

}

}
```

The screenshot shows the IntelliJ IDEA interface. The left pane displays the project structure for 'Ass6' with files like Armstrong, GCD_LCM, Knapsack, MagicNumberCheck, Main, .gitignore, and Ass6.iml. The right pane shows the code editor with MagicNumberCheck.java open. The code reads a number from standard input, reduces it to a single digit by summing its digits, and then checks if the original number equals this sum. The code editor has several red highlights on lines 11, 12, 13, 14, 15, 16, 17, 18, and 19. Below the code editor is the 'Threads & Variables' and 'Console' tabs. The 'Console' tab shows the output of running the program with the input '119'. It prints 'Enter the number to be checked.', then '119', followed by '119 is not a Magic Number.', and finally 'Process finished with exit code 0'. The bottom status bar indicates the file is 111 lines long, uses CRLF line endings, is in UTF-8 encoding, and has 4 spaces per indentation.

```

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;

    // Keep reducing num until it's a single digit
    while (num > 9) {
        int sum = num; // Initialize sum to the current num
        int s = 0;

        // Sum the digits of num
        while (sum > 0) {
            s = s + (sum % 10); // Add the last digit to s
            sum = sum / 10; // Move to the next digit
        }

        num = s; // Update num to the sum of digits
    }
}

```

Now the loop is executed entirely and prints the valid answer.

Code 5: Merge Sort

How many errors are there in the program?

Mention the errors you have identified.

1. Incorrect Array Slicing:

- In the `mergeSort` method, the lines `int[] left = leftHalf(array + 1);` and `int[] right = rightHalf(array - 1);` incorrectly attempt to slice the array. Instead, you should pass the whole array to the `leftHalf` and `rightHalf` methods.

Correction: Pass the entire array to the `leftHalf` and `rightHalf` methods without modifying the array itself.

2. Incorrect Indexing:

- The increment operations `left++` and `right--` in the `merge` method call are invalid because they are not intended to increment array references. Instead, you should pass the entire left and right arrays as they are.

Correction: Simply use `merge(array, left, right);` without any increment or decrement.

3. Size of the Left Array:

- The `leftHalf` method is returning an array that does not include the middle element when the length of the original array is odd. It should handle this case

correctly to ensure the split is accurate.

Correction: Update the size calculation in the `leftHalf` method to consider both even and odd lengths correctly.

4. Merge Result Array Not Initialized:

- The `merge` method should operate on the result array passed into it, but it needs to ensure that the result array is the same size as the original array being sorted.

5. Array Indexing in the Merge Method:

- The loop in the `merge` method does not properly handle the condition when one of the arrays is exhausted.

How many breakpoints do you need to fix those errors?

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

1. Before the `mergeSort` call:

- Set a breakpoint at the beginning of the `mergeSort` method to inspect the initial state of the array before sorting begins.

2. Before the left and right halves are split:

- Set a breakpoint after splitting the array into left and right halves. This will help verify that the splitting logic works correctly.

3. Inside the merge method:

- Set a breakpoint at the beginning of the `merge` method to check the contents of the left and right arrays being merged.

4. Inside the merge loop:

- Set a breakpoint inside the for loop of the `merge` method to see how elements from the left and right arrays are being compared and merged.

```
// Places the elements of the given array into sorted order
// using the merge sort algorithm.
// post: array is in sorted (nondecreasing) order
3 usages
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.
```

Build Output

- Ass6: build failed At 19-10-2024 20:02 with 4 error 3 sec, 150 ms
- MergeSort.java src 4 errors
 - bad operand types for binary operator '+'::20
first type: int[]
second type: int
 - bad operand types for binary operator '*'::21
first type: int[]
second type: int
 - bad operand type int[] for unary operator '++'::28
first type: int[]
 - bad operand type int[] for unary operator '--'::28
first type: int[]

Here, `array + 1` cannot be done.

Steps to Fix the Errors Identified in the Code Fragment:

1. **Correct Array Slicing:**
 - Modify the calls to `leftHalf` and `rightHalf` to pass the original array instead of attempting to increment or decrement it.
2. **Remove Incorrect Increment Operations:**
 - Change the merge call to just `merge(array, left, right);` without using `++` or `--`.
3. **Fix Left Half Size Calculation:**
 - Update the size calculation in `leftHalf` to correctly account for odd-length arrays.
4. **Ensure Merge Logic Works:**
 - Check the merge logic to ensure that it correctly handles cases where one of the halves is exhausted before the other.
5. **Test the Functionality:**
 - After making the changes, run the program with various inputs to confirm that it sorts the array correctly.

```
import java.util.*;

public class MergeSort {

    public static void main(String[] args) {

        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};

        System.out.println("before: " + Arrays.toString(list));

        mergeSort(list);

        System.out.println("after: " + Arrays.toString(list));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.

    // post: array is in sorted (nondecreasing) order

    public static void mergeSort(int[] array) {
```

```

if (array.length > 1) {

    // split array into two halves

    int[] left = leftHalf(array);

    int[] right = rightHalf(array);

    // recursively sort the two halves

    mergeSort(left);

    mergeSort(right);

    // merge the sorted halves into a sorted whole

    merge(array, left, right);

}

}

// Returns the first half of the given array.

public static int[] leftHalf(int[] array) {

    int size1 = (array.length + 1) / 2; // Handle odd lengths

    int[] left = new int[size1];

    for (int i = 0; i < size1; i++) {

        left[i] = array[i];

    }

    return left;

}

}

// Returns the second half of the given array.


```

```

public static int[] rightHalf(int[] array) {

    int size1 = array.length / 2;

    int size2 = array.length - size1;

    int[] right = new int[size2];

    for (int i = 0; i < size2; i++) {

        right[i] = array[i + size1];

    }

    return right;

}

// Merges the given left and right arrays into the given
// result array.

// pre : result is empty; left/right are sorted
// post: result contains result of merging sorted lists;

public static void merge(int[] result, int[] left, int[] right) {

    int i1 = 0; // index into left array

    int i2 = 0; // index into right array


    for (int i = 0; i < result.length; i++) {

        if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {

            result[i] = left[i1]; // take from left

            i1++;

        } else {

            result[i] = right[i2]; // take from right

            i2++;

        }

    }

}

```

```

    }
}

}
}

```

```

src\Ass6.java  Armstrong.java  GCD_LCM.java  Knapsack.java  MagicNumberCheck.java  MergeSort.java
1 import java.util.*;
2
3 public class MergeSort {
4     public static void main(String[] args) {
5         int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
6         System.out.println("before: " + Arrays.toString(list));
7         mergeSort(list);
8         System.out.println("after: " + Arrays.toString(list));
9     }
10
11     3 usages
12     public static void mergeSort(int[] array) {
13         if (array.length > 1) {
14             // split array into two halves
15             int[] left = leftHalf(array); // Corrected
16             int[] right = rightHalf(array); // Corrected
17
18             // recursively sort the two halves
19             mergeSort(left);
20             mergeSort(right);
21         }
22     }
23 }

```

The IDE interface shows the project structure for 'Ass6' with files like Armstrong.java, GCD_LCM.java, Knapsack.java, MagicNumberCheck.java, and MergeSort.java. The MergeSort.java file is open in the editor. The 'Debug' tab is selected, showing a connection to a target VM (transport: 'socket'). The 'Console' tab shows the output of the program execution:

```

Connected to the target VM, address: '127.0.0.1:50276', transport: 'socket'
before: [34, 32, 67, 76, 23, 41, 58, 85]
after: [34, 23, 32, 41, 58, 67, 76, 85]
Disconnected from the target VM, address: '127.0.0.1:50276', transport: 'socket'
Process finished with exit code 0

```

Code 6: Multiply Matrices

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

1. Matrix Indexing Errors:

- The multiplication logic uses incorrect indexing (`first[c-1][c-k]` and `second[k-1][k-d]`), which will lead to `ArrayIndexOutOfBoundsException` when `c` or `k` is 0.

2. Resetting Sum:

- The variable `sum` should be reset to 0 before accumulating values for each element in the resulting matrix. While this is done, it is better placed right before starting the inner for loop.

3. Input for the Second Matrix:

- The prompt message for the second matrix asks for the same input as the first matrix, which can be misleading.

How many breakpoints do you need to fix those errors?

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

1. After reading the first matrix:
 - o Check if the matrix is populated correctly.
2. After reading the second matrix:
 - o Verify if the second matrix is also populated correctly.
3. Before entering the matrix multiplication logic:
 - o Inspect the dimensions of the matrices to ensure they can be multiplied.
4. Inside the multiplication loop:
 - o Verify the computation of each element in the resulting matrix.

```

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++ )
            {
                multiply[c][d] += second[c][k] * matrix[c][k];
            }
        }
    }
}

```

Could not debug successfully

Steps to Fix the Errors:

1. **Correct Indexing:**
 - o Change the indexing in the multiplication loop to use the current indexes directly.
2. **Reset the Sum:**
 - o Move the `sum = 0;` to the correct position.
3. **Improve Input Prompts:**
 - o Clarify the prompts for entering the second matrix.

```
import java.util.Scanner;
```

```
class MatrixMultiplication {
```

```
public static void main(String args[]) {  
    int m, n, p, q, sum, c, d, k;  
  
    Scanner in = new Scanner(System.in);  
  
    System.out.println("Enter the number of rows and columns of first matrix");  
  
    m = in.nextInt();  
  
    n = in.nextInt();  
  
    int first[][] = new int[m][n];  
  
    System.out.println("Enter the elements of first matrix");  
  
    for (c = 0; c < m; c++)  
  
        for (d = 0; d < n; d++)  
  
            first[c][d] = in.nextInt();  
  
    System.out.println("Enter the number of rows and columns of second matrix");  
  
    p = in.nextInt();  
  
    q = in.nextInt();  
  
    if (n != p) {  
  
        System.out.println("Matrices with entered orders can't be multiplied with each other.");  
  
    } else {  
  
        int second[][] = new int[p][q];  
  
        int multiply[][] = new int[m][q];  
  
        System.out.println("Enter the elements of second matrix");  
    }  
}
```

```

for (c = 0; c < p; c++)
    for (d = 0; d < q; d++)
        second[c][d] = in.nextInt();

// Matrix multiplication

for (c = 0; c < m; c++) {
    for (d = 0; d < q; d++) {
        sum = 0; // Reset sum for each element in the resulting matrix
        for (k = 0; k < n; k++) { // Use n here since it corresponds to columns of first and
rows of second
            sum += first[c][k] * second[k][d]; // Correct indexing
        }
        multiply[c][d] = sum;
    }
}

// Print the product of the matrices

System.out.println("Product of entered matrices:-");

for (c = 0; c < m; c++) {
    for (d = 0; d < q; d++)
        System.out.print(multiply[c][d] + "\t");
    System.out.print("\n");
}

in.close(); // Close the scanner

```

```
}
```

The screenshot shows the IntelliJ IDEA interface with the project 'Ass6' open. The 'src' directory contains several Java files: Armstrong, GCD_LCM, Knapsack, MagicNumberCheck, Main, MatrixMultiplication, and MergeSort. The 'MatrixMultiplication.java' file is currently selected and shown in the editor. The code implements matrix multiplication. The 'Debug' tool window at the bottom shows the execution of the program. It prompts for matrix dimensions and elements, then calculates the product. The output shows the result of multiplying a 3x0 matrix by a 7x0 matrix, resulting in a 7x0 matrix. The status bar indicates the process finished with exit code 0.

```
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 (int c = 0; c < m; c++)
    for (int 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();

Product of entered matrices:-
3 0
7 0
```

Successfully debugged

Code 7: Quadratic Probing

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

1. **Syntax Errors:**
 - o In the `insert` and `get` methods, the code for updating the index `i` was incorrect due to improper spacing. The expression was written as `i += (i + h / h--) % maxSize;`, which caused a compilation error. The correct syntax is `i += (h * h++) % maxSize;`, ensuring that `i` is updated properly for probing.
2. **Incorrect Loop Condition:**
 - o In the `get` method, the while loop condition only checks if `keys[i]` is not null. It should also check if the key matches to avoid accessing uninitialized slots. The condition should be `while (keys[i] != null && !keys[i].equals(key))`.
3. **Unnecessary Reinitialization in `makeEmpty`:**
 - o The `makeEmpty` method reinitialized the `keys` and `vals` arrays, which was unnecessary. Instead, it should only reset `currentSize` to 0 to clear the hash table while retaining the allocated space for future inserts.

4. Off-by-One Error in `remove` Method:

- The logic for rehashing elements after a deletion had a flaw where it did not properly handle the indices for rehashing. This could lead to incorrect behavior when re-inserting keys.

5. Uninitialized Variables:

- The variable `tmp` in the `insert` method was used for collision management but was not correctly utilized, which could result in infinite loops or incorrect behavior when handling collisions.

How many breakpoints do you need to fix those errors?

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

1. After Inserting a Key-Value Pair:

- To verify that a new entry is correctly added.

2. When Checking for Existing Keys:

- To ensure the program correctly identifies whether a key already exists in the hash table.

3. At the Beginning of the `get` Method:

- To track the process of searching for a given key.

4. During Rehashing in the `remove` Method:

- To monitor the logic involved in rehashing after a deletion, ensuring all keys are properly repositioned.

```
109     /** Function to print HashTable */
110     1 usage
111     public void printHashTable()
112     {
113         System.out.println("\nHash Table: ");
114         for (int i = 0; i < maxsize; i++)  maxSize: 5
115             if (keys[i] != null)
116                 System.out.println(keys[i] + " " + vals[i]);
117         System.out.println();
118     }
```

Threads & Variables Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

main@1 in g.main: RUNNING

printHashTable@113, QuadraticProbingHashTableTest@main:165

this = (QuadraticProbingHashTable@654)

vals = [String[5]@656] [null, null, null, null, "computer"]

keys = [String[5]@655] [null, null, null, null, "c"]

maxSize = 5

Here it added only the first key-value pair and not the others and were kept null

Steps to Fix Errors:

To correct the identified issues:

1. **Fix Syntax Errors:**
 - Update the incorrect index update lines in both the `insert` and `get` methods.
2. **Update Loop Conditions:**
 - Change the while loop condition in the `get` method to ensure it checks for both null values and key equality.
3. **Simplify `makeEmpty`:**
 - Remove the unnecessary reinitialization of the arrays in the `makeEmpty` method.
4. **Review Rehashing Logic:**
 - Ensure that the rehashing logic in the `remove` method is implemented correctly, particularly with how indices are updated after a deletion.
5. **Test Changes:**
 - After making these corrections, run the program to confirm it functions correctly without errors.

```
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;  
  
    // No need to reinitialize keys and vals arrays  
  
}
```

```
/** 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) {
```

```
    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; // Update the value
```

```
            return;
```

```
}
```

```
        i = (i + h * h) % maxSize; // Corrected the update logic
```

```
        h++;
```

```

} while (i != tmp);

}

/** Function to get value for a given key **/

public String get(String key) {

    int i = hash(key), h = 1;

    while (keys[i] != null) {

        if (keys[i].equals(key)) {

            return vals[i];

        }

        i = (i + h * h) % maxSize; // Corrected the update logic

        h++;

    }

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

```

import java.util.Scanner;

/** Class QuadraticProbingHashTable */
2 usages
class QuadraticProbingHashTable {
    7 usages
    private int currentSize, maxSize;
    13 usages
    private String[] keys;
    8 usages
    private String[] vals;
    /** Constructor */
    1 usage
    public QuadraticProbingHashTable(int capacity) {
        11 currentSize = 0;
        maxSize = capacity;
    }
}

```

Successfully debugged

Code 8: Sorting Array

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

- 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 or use an underscore without spaces.
- Loop Condition Error:** The condition in the outer for loop is incorrect. It uses `i >= n`, which will cause the loop to never execute. It should be `i < n`.
- Semicolon after the Loop:** There is a semicolon after the outer for loop (`for (int i = 0; i >= n; i++) ;`). This semicolon ends the loop prematurely, and the inner loop is not associated with the outer loop.
- Sorting Logic:** The comparison in the inner loop should use `<` instead of `<=` for a proper ascending sort. Using `<=` can cause unnecessary swaps.

How many breakpoints do you need to fix those errors?

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

1. Class declaration (to correct the class name).
2. First for loop condition (to correct the loop condition).
3. Inner loop (to remove the semicolon).
4. Inside the inner loop where the comparison occurs (to change `<=` to `<`).
5. Printing logic (to handle the last comma issue).

The screenshot shows the IntelliJ IDEA interface with the project structure on the left. The main editor window displays the code for `Ascending_Order.java`. Several lines of code are underlined with red, indicating errors. The build output panel at the bottom shows a single error: `C:\Users\Hetzvi\Downloads\Ass6\src\Ascending_Order.java:3:23 java: '{' expected`. This error is highlighted in the code editor.

```

1 // sorting the array in ascending order
2 import java.util.Scanner;
3 public class Ascending_Order
4 {
5     public static void main(String[] args)
6     {
7         int n, temp;
8         Scanner s = new Scanner(System.in);
9         System.out.print("Enter no. of elements you want in array:");
10        n = s.nextInt();
11        int a[] = new int[n];
12        System.out.println("Enter all the elements:");
13        for (int i = 0; i < n; i++)
14        {
15            a[i] = s.nextInt();
16        }
17        for (int i = 0; i >= n; i++)
18    }

```

Could not complete debugging because of space in class name.

Steps Taken to Fix the Errors:

1. **Change Class Name:** Rename `Ascending _Order` to `AscendingOrder`.
2. **Correct Loop Condition:** Change `i >= n` to `i < n` in the outer loop.
3. **Remove Semicolon:** Delete the semicolon after the outer for loop declaration.
4. **Modify Sorting Logic:** Change `if (a[i] <= a[j])` to `if (a[i] < a[j])` for correct ascending order sorting.
5. **Adjust Printing Logic:** Update the printing logic to avoid a trailing comma after the last element.

```

import java.util.Scanner;

public class AscendingOrder {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);

```

```

System.out.print("Enter no. of elements you want in array: ");
n = s.nextInt();

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

// Sorting 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 comparison for ascending order
            // Swap elements
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
}

// Printing the sorted array
System.out.print("Ascending Order: ");
for (int i = 0; i < n; i++) { // Adjusted printing loop
    System.out.print(a[i]);
    if (i < n - 1) { // Print comma for all but the last element
        System.out.print(", ");
    }
}
}
}

```

Successfully debugged

Code 9: Stack Implementation

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

- Push Logic:** The `push` method incorrectly decrements `top` before assigning the value. This causes the first value to be stored in an invalid index.
- Pop Logic:** The `pop` method incorrectly increments `top` instead of decrementing it, leading to incorrect stack behavior.
- Display Logic:** The for loop condition in the `display` method is `i > top`, which prevents any elements from being displayed. It should be `i <= top`.
- Class Naming Convention:** The class name `StackMethods` does not follow standard naming conventions in Java. While not a functional error, it is good practice to use a more concise name like `Stack`.

How many breakpoints do you need to fix those errors?

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

1. Class declaration line (`public class StackMethods`).

- Reason: Check for any naming or compilation issues.
- Input of value in the `push` method (`stack[top] = value;`).
 - Reason: Verify that the `top` index is being used correctly before assigning the value. - Pop method (`if (!isEmpty())`).
 - Reason: Check the condition for popping and confirm that it allows for correct execution. - Inside the `display` method for loop condition (`for (int i = 0; i > top; i++)`).
 - Reason: Ensure the loop executes correctly and allows for displaying elements. - Before the pop operation (`System.out.println("Popped: " + stack[top]);`).
 - Reason: Confirm that the value to be popped is correctly accessed.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows a tree structure of Java files under the `src` directory, including `Armstrong`, `Ascending_Order`, `GCD_LCM`, `Knapsack`, `MagicNumberCheck`, `Main`, `MatrixMultiplication`, `MergeSort`, and `StackReviseDemo.java`.
- Code Editor:** Displays the `StackMethods` class definition. The code includes comments, imports, and a constructor. A red error indicator is present on the first line of the class definition.
- Build Output:** Shows a failed build with one error: `java: class StackMethods is public, should be declared in a file named StackMethods.java`.
- Status Bar:** Shows the path `C:\Users\Hetzvi\Downloads\Ass6>src>StackReviseDemo.java>StackMethods` and file statistics: 4/8 CRLF, UTF-8, 4 spaces.

Could not build the code.

Steps to Fix the Identified Errors:

1. Fix Push Logic:

Change the `push` method to increment `top` before assigning the value:

```
top++;
stack[top] = value;
```

2. Fix Pop Logic:

Modify the `pop` method to decrement `top` instead of incrementing it:
`top--;`

3. Fix Display Logic:

Change the loop condition in the `display` method from `i > top` to `i <= top`:
`for (int i = 0; i <= top; i++)`

4. Update Class Naming:

Rename `StackMethods` to `Stack` for better naming convention:

```
class Stack { ... }
```

```
import java.util.Arrays;

class Stack {
    private int top; // Index of the top element
    private int size; // Maximum size of the stack
    private int[] stack; // Array to hold stack elements

    // Constructor to initialize stack
    public Stack(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1; // Indicates an empty stack
    }

    // Method to push a value onto 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
            stack[top] = value; // Assign value to the top position
        }
    }

    // Method to pop a value from the stack
    public void pop() {
        if (!isEmpty()) {
```


}

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure for 'Ass6' with files like Armstrong, Ascending_Order, GCD_LCM, Knapsack, MagicNumberCheck, Main, MatrixMultiplication, MergeSort, StackReviseDemo.java, and Armstrong.java. The right pane shows the code for StackReviseDemo.java:

```
public void display() {
    if (isEmpty()) {
        System.out.println("Stack is empty.");
        return;
    }
    System.out.print("Stack elements: ");
    for (int i = 0; i <= top; i++) { // Loop until top
        System.out.print(stack[i] + " ");
    }
    System.out.println();
}
public class StackReviseDemo {
    public static void main(String[] args) {
        Stack newStack = new Stack(arraySize: 5);
    }
}
```

The 'Threads & Variables' tool window at the bottom shows a running thread named 'main'@1 in g.main. The variable 'display@46' is expanded, showing 'this = (Stack@490)' and 'stack = [int@5]@492 [10, 1, 50, 20, 90]' with 'top = 4'. The status bar at the bottom indicates '46.1 CPU: 100% LITE: 9 A spaces'.

Successfully debugged push operation

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure for 'Ass6'. The right pane shows the code for StackReviseDemo.java, specifically the 'display' method:

```
    top--; // Decrement top to remove the top element
}
// Method to check if the stack is empty
public boolean isEmpty() {
    return top == -1;
}
// Method to display stack elements
public void display() {
}
```

The 'Threads & Variables' tool window at the bottom shows the stack elements being popped: 'Popped: 90', 'Popped: 20', 'Popped: 50', 'Popped: 1'. The status bar at the bottom indicates '46.1 CPU: 100% LITE: 9 A spaces'.

Successfully debugged pop operation

Code 10: Tower of Hanoi

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

- **Infinite Recursion:**
 - **Line 12:** The call `doTowers(topN++, inter--, from + 1, to + 1)` is incorrect because it modifies the parameters inappropriately.
 - `topN++` will increment `topN` after the method call, which leads to infinite recursion because it always tries to call with a non-decreasing value of `topN`.
 - `inter--` is also incorrect since it attempts to decrement a char, which doesn't logically represent the movement of disks.
 - `from + 1` and `to + 1` attempt to increment characters, which doesn't produce the desired result.
- **Output Message Issue:**
 - The output messages should clarify which disk is being moved and from where to where, but currently, it does not specify the total number of disks being moved.

How many breakpoints do you need to fix those errors?

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

1. **Breakpoint 1:** Before the first if statement
 - **Line 10:** To check the entry condition of the recursion.
2. **Breakpoint 2:** Inside the else block before the first recursive call
 - **Line 12:** To observe parameters passed to the first recursive call.
3. **Breakpoint 3:** Just before the `println` statement in the else block
 - **Line 14:** To see which disk is being moved.
4. **Breakpoint 4:** Inside the second recursive call
 - **Line 15:** To check parameters passed to the second recursive call.

```

    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);
    }
}

```

Could not compile successfully.

Steps to Fix the Errors:

To fix the identified errors:

1. Correct the Recursive Calls:

- Change `doTowers(topN++, inter--, from + 1, to + 1)` to `doTowers(topN - 1, inter, from, to)` for the recursive call after moving the smaller disks. This maintains the correct logic of moving `topN - 1` disks to the intermediate peg before moving the largest disk.

2. Maintain Character Values:

- `from`, `to`, and `inter` should be used without incrementing or decrementing since they are supposed to represent character labels (A, B, C).

3. Reformat the Output:

- Adjust the print statements for better clarity if desired.

```

// Tower of Hanoi
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3; // Number of disks
        doTowers(nDisks, 'A', 'B', 'C'); // A: Source, B: Auxiliary, C: Destination
    }
}

```

```
public static void doTowers(int topN, char from, char inter, char to) {
```

```

if (topN == 1) {
    System.out.println("Disk 1 from " + from + " to " + to);
} else {
    // Move topN-1 disks from source to auxiliary
    doTowers(topN - 1, from, to, inter);
    // Move the largest disk from source to destination
    System.out.println("Disk " + topN + " from " + from + " to " + to);
    // Move the topN-1 disks from auxiliary to destination
    doTowers(topN - 1, inter, from, to);
}
}
}
}

```

```

1 // Tower of Hanoi
2 public class MainClass {
3     public static void main(String[] args) { args: []
4         int nDisks = 3; // Number of disks nDisks: 3
5         doTowers(nDisks, from:'A', inter:'B', to:'C'); // A: Source, B: Auxiliary, C: Destination nDisks: 3
6     }
7
8     3 usages
9
10    public static void doTowers(int topN, char from, char inter, char to) {
11        if (topN == 1) {
12            System.out.println("Disk 1 from " + from + " to " + to);
13        } else {
14            // Move topN-1 disks from source to auxiliary
15            doTowers( topN - 1, from, inter, to);
16            // Move the largest disk from source to destination
17            doTowers( topN - 1, inter, from, to);
18        }
19    }
20
21 }
22
23 }
24
25 }

```

Threads & Variables Console Breakpoints Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

main:6, MainClass args = String[0]@491 nDisks = 3

Successfully Debugged

The screenshot shows the IntelliJ IDEA interface with the project 'Ass6' open. The 'src' directory contains several Java files: Armstrong, Ascending_Order, GCD_LCM, Knapsack, MagicNumberCheck, Main, MainClass, MatrixMultiplication, MergeSort, and StackReviseDemo.java. The 'MainClass.java' file is currently selected and displayed in the editor. The code implements the Tower of Hanoi algorithm:

```
// Tower of Hanoi
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3; // Number of disks
        doTowers(nDisks, 'A', 'B', 'C'); // A: Source, B: Auxiliary, C: Destination
    }

    public static void doTowers(int topN, char from, char inter, char to) {
        if (topN == 1) {
            System.out.println("Disk 1 from " + from + " to " + to);
        } else {
            // Move topN-1 disks from source to auxiliary
            doTowers( topN - 1, from, to, inter );
            // Move the largest disk from source to destination
        }
    }
}
```

The 'Threads & Variables' tool window shows the execution flow, indicating steps like 'Disk 1 from A to C', 'Disk 2 from A to B', etc. The 'Console' tab shows the output of the program, which includes the moves and a final message: 'Process finished with exit code 0'. The status bar at the bottom right indicates '6.1 _CBLE_ LITE-R 4 spaces'.

Que3 : Static analysis tool (SpotBugs)

Error code:

```
67     @Override
68     public void run() {
69         try {
70             // read
71             BufferedReader br = new BufferedReader(new InputStreamReader(server.sk.getInputStream()));
72
73             // writing
74             pw = new PrintWriter(server.sk.getOutputStream(), autoFlush: true);
75             name = br.readLine();
76             server.broadcast("**[" + name + "] Entered**");
77
78             String data;
```

Found reliance on default encoding: new java.io.InputStreamReader(InputStream)

Class: ServerThread () line 71 Internationalization (Dubious method used)
DM DEFAULT_ENCODING (Reliance on default encoding)

Method: run (ServerThread run())

Notes: Called method new

Correctness:

```

    //addr = InetAddress.getByName("192.168.43.1");
    ...
    server = new ServerSocket( port: 1234, backlog: 50,addr);
    System.out.println("\n Waiting for Client connection");
    SocketClient.main( args: null);
    while(true) {
        sk = server.accept();
        System.out.println(sk.getInetAddress() + " connect");
        ...
    }
}

```

SpotBugs

- ChitChat-master (found 6 bug items in 3 classes)
 - Internationalization (4 items)
 - Dubious method used (4 items)
 - Reliance on default encoding (4 items)
 - Found reliance on default encoding in Server
 - Found reliance on default encoding in Server
 - Found reliance on default encoding in Sock
 - Found reliance on default encoding in Sock
 - Correctness (1 item)
 - Null pointer dereference (1 item)
 - Method call passes null to a non-null parameter
 - Nul passed for non-null parameter of Sock
 - Multithreaded correctness (1 item)

Preview SocketServer.java:

```

    ...
    server = new ServerSocket( port: 1234, backlog: 50,addr);
    System.out.println("\n Waiting for Client connection");
    SocketClient.main( args: null);
    while(true) {
        sk = server.accept();
        System.out.println(sk.getInetAddress() + " connect");
        ...
    }
}

```

Method call passes null to a non-null parameter

This method passes a null value as the parameter of a method which must be non-null. Either this parameter has been explicitly marked as @Nonnull, or analysis has determined that this parameter is always dereferenced.

Multithreaded correctness:

```

    ...
    st.start();
}
} catch(IOException e) {
    System.out.println(e + "-> ServerSocket failed");
}
}

```

SpotBugs

- ChitChat-master (found 6 bug items in 3 classes)
 - Internationalization (4 items)
 - Dubious method used (4 items)
 - Reliance on default encoding (4 items)
 - Found reliance on default encoding in Server
 - Found reliance on default encoding in Server
 - Found reliance on default encoding in Sock
 - Found reliance on default encoding in Sock
 - Correctness (1 item)
 - Null pointer dereference (1 item)
 - Method call passes null to a non-null parameter
 - Nul passed for non-null parameter of Sock
 - Multithreaded correctness (1 item)
 - Constructor invokes Thread.start() (1 item)
 - Constructor invokes Thread.start() (1 item)
 - new SocketServer() invokes ServerThreads

Preview SocketServer.java:

```

    ...
    st.start();
}
} catch(IOException e) {
    System.out.println(e + "-> ServerSocket failed");
}
}

```

new SocketServer() invokes ServerThread.start()

The constructor starts a thread. This is likely to be wrong if the class is ever extended/subclassed, since the thread will be started before the subclass constructor is started.