Lab 7 : Software Engineering

202201512 – Chirag Patel

I. PROGRAM INSPECTION:

An Error Checklist for Inspections

An important part of the inspection process is the use of a checklist to examine the program for common errors. Unfortunately, some checklists concentrate more on issues of style than on errors (for example, “Are comments accurate and meaningful?” and “Are if- else, code blocks, and do..whilgroups aligned?”), and the error checks are too nebulous to be useful (such as “Does the code meetthe design requirements?”). The checklist in this section was compiled after many years of study ofsoftware errors. The checklist is largely language independent, meaning that most of the errors canoccur with any programming language. You may wish to supplement this list with errors peculiar tyour programming language and with errors detected after using the inspection process.

Category A: Data Reference Errors

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

2. For all array references, is each subscript value within the defined bounds ofthe correspondingdimension?

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.

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 toan 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.

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 aremade 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.

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.

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 a bit-string argument to a subroutine.

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.

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

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?

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

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?

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.

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.

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

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

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.

Category C: Computation Errors

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

2. Are there any mixed-mode computations? An example is the addition of a floating-point variable to an integer variable. Such occurrences are not necessarily errors, but they should beexplored carefully to ensure that the language's conversion rules are understood. Consider thefollowing Java snippet showing the rounding error that can occur when working with integers:

```
int x = 1;

int y = 2;

int z = 0;

z = x/y;

System.out.println (&amp;quot;z = &amp;quot; + z);
```

OUTPUT:

z = 0

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

4. Is the data type of the target variable of an assignment 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? Thatis, the end result may appear to have valid value, but an intermediate result might be too big ortoo small for the programming language's data types.

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

7. If the underlying machine represents variables in base-2 form, are there any sequences of the resulting inaccuracy? That is, 10 x 0.1 is rarely equal to 1.0 on a binary machine.

8. Where applicable, can the value of a variable go outside the meaningful range? For example, statements assigning a value to the variable PROBABILITY might be checked to ensure that the assigned value will always be positive and not greater than

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

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

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?

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

3. Are the comparison operators correct? Programmers frequently confuse such relations as atmost, at least, greater than, not less than, less than or equal.

4. Does each Boolean expression state what it is supposed to state? Programmers often makemistakes when writing logical expressions involving and, or, and not.

5. Are the operands of a Boolean operator Boolean? Have comparison and Boolean operatorsbeen erroneously mixed together? This represents another frequent class of mistakes.Examples of a few typical mistakes are illustrated here. If you want to determine whether i isbetween 2 and 10, the expression 2<i<10 is incorrect; instead, it should be (2<i) && (i<10). Ifyou want to determine whether i is greater than x or y, i>x||y is incorrect; instead, it should be (i>x)||(i>y). If you want to compare three numbers for equality, if(a==b==c) does something quite different. If you want to test the mathematical relation x>y>z, the correct expression is(x>y)&&(y>z).

6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine? This is an occasional source of errors because of truncation and base-2 approximations of base-10 numbers.

7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct? That is, if you see an expression such as (if((a==2) && (b==2) || (c==3)), is it well understood whether the and or the or is performed first?

8. Does the way in which the compiler evaluates Boolean expressions affect the program? For instance, the statement if((x==0 && (x/y)>z) may be acceptable for compilers that end the test as soon as one side of an and is false, but may cause a division-by-zero error with other compilers.

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? For example, in the statement

GO TO (200, 300, 400), i

will i always have the value of 1, 2, or 3?

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

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

4. Is it possible that, because of the conditions upon entry, a loop will never execute? If so, does this represent an over- sight? For instance, if you had the following loops headed by the

following statements:

for (i==x ; i<=z; i++) {

...

}

while (NOTFOUND) {

...

}

what happens if NOTFOUND is initially false or if x is greater than z?

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? For example, for the psuedo-code loop headed by DO I=1 to TABLESIZE WHILE (NOTFOUND)

what happens if NOTFOUND never becomes false?

6. Are there any off-by-one errors, such as one too many or too few iterations? This is a common error in zero-based loops. You will often forget to count “0” as

a number. For example, if you want to create Java code for a loop that counted to 10, the following would be wrong, as it counts to 11:

for (int i=0; i<=10;i++) {

System.out.println(i);

}

Correct, the loop is iterated 10 times:

for (int i=0; i <=9;i++) {

System.out.println(i);

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.

8. Are there any non-exhaustive decisions? For instance, if an input parameter's expected values are 1, 2, or 3, does the logic assume that it must be 3 if it is not 1 or 2? If so, is the assumption valid?

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?

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

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?

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

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

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

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

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

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

Category G: Input / Output Errors

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

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

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

4. Have all files been opened before use?

5. Have all files been closed after use?

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

7. Are I/O error conditions handled correctly?

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

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.

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

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 usesthat impede code optimization.

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

5. Is there a function missing from the program?

Program Inspection: (Submit the answers of following questions for each code fragment)

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

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

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

4. Is the program inspection technique is worth applicable?

II. CODE DEBUGGING: Debugging is the process of localizing, analyzing, and removing suspectederrors in the code (Java code given in the .zip file)

Instructions (Use Eclipse/Netbeans IDE, GDB Debugger)

• Open a NEW PROJECT. Select Java/C++ application. Give suitable name to the file.

• Click on the source file in the left panel. Click on NEW in the pull down menu.

• Select main Java/C++ file.

• Build and Run the project.

• Set a toggle breakpoint to halt execution at a certain line or function

• Display values of variables and expressions

• Step through the code one instruction at a time

• Run the program from the start or continue after a break in the execution

• Do a backtrace to see who has called whom to get to where you are

• Quit debugging.

Debugging: (Submit the answers of following questions for each code fragment)

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

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

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

3. Submit your complete executable code?

Static Analysis Tools

Choose a static analysis tool (in Java, Python, C, C++) in any programming language of your interest andidentify the defects. You can also choose your own code fragment from GitHub (more than 2000 LOC)in any programming language to perform static analysis. Submit your results in the .xls or .jpg format only.

Q1 Armstrong

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

Program Inspection

1. Errors: a. Error 1: In the while loop, the remainder is calculated incorrectly. It should be remainder = num % 10; instead of remainder = num / 10;. b. Error 2: The class Armstrong is missing a closing bracket '}'.
2. Effective Categories:
    - Category B: Data Declaration Errors.
    - Category E: Control Flow Errors.
3. Unidentified Issues:
    - The program inspection didn't find possible problems like integer overflow.

4. Applicability:
    - Program inspection can help catch syntax and some logic errors, but it doesn't check if the program's logic is correct.

Debugging

1. Errors: a. Error 1: In the while loop, change the calculation of remainder to remainder = num % 10; to calculate it correctly.
2. Breakpoints Needed:
    - At least one breakpoint is needed to fix these errors.

Corrected Code

```
public 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 & gt; 0) {
            remainder = num % 10; // Corrected calculation of remainder
            check = check + (int) Math.pow(remainder, 3);
            num = num / 10; // Corrected calculation of num
        }
    if (check == n)
        System.out.println(n + & quot; is an Armstrong Number & quot;);
    else
        System.out.println(n + & quot; is not an Armstrong Number &
quot;);
    }
}
```

Q2 : LCM and GCD

```
//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 LCM of two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

Program Inspection

1. Errors: a. Error 1: In the gcd method, the condition for the while loop is wrong. It should be while(b != 0) instead of while(a % b == 0).
2. Effective Category:
    - Category B: Semantic Errors.
3. Unidentified Issues:
    - The program inspection did not catch possible problems like integer overflow or mistakes in the lcm method.
4. Applicability:
    - Program inspection can help find syntax and some logic errors, but it doesn't confirm if the mathematical logic in the gcd and lcm methods is correct.

Debugging:

1. Errors: a. Error 1: In the gcd method, change the while loop condition to while(b != 0) to correctly determine the greatest common divisor.
2. Breakpoints Needed:
    - At least one breakpoint is needed to help fix these errors.

Corrected Code

```
public class GCD_LCM {
```

```
    static int gcd(int x, int y) {
        int r = 0, a, b;
        a = (x &amp;gt; y) ?y: x; // a is greater number
        b = (x &amp;lt; y) ?x: y; // b is smaller number
        r = b;
        while (a % b != 0) // Corrected loop condition
        {
            r = a % b;
            a = b;
            b = r;
        }
        return r;
    }
    static int lcm(int x, int y) {
        int a;
        a = (x &amp;gt; y) ?x: y; // a is greater number
        while (true) {
            if (a % x != 0 &amp;amp;&amp;amp; a % y != 0)
            return a;
            ++a;
        }
    }
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println(&amp;quot;Enter the two numbers: &amp;quot;);
        int x = input.nextInt();
        int y = input.nextInt();
        System.out.println(&amp;quot;The GCD of two numbers is: &amp;quot; +
gcd(x, y));
        System.out.println(&amp;quot;The LCM of two numbers is: &amp;quot; +
lcm(x, y));
        input.close();
    }
}
```

Q3 Knapsack

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

Program Inspection

1. Errors:
    a. Error 1: In the for loop header, there is a post-increment operator (`n++`) used instead of just incrementing `n` by one (`n++` should be `n+1`).
    b. Error 2: The indexing of arrays should be from `0` to `N`, but it starts from `1` to `N`. In Java, arrays are zero-indexed.

c. Error 3: In the option2 calculation, the code is using the item's profit and weight at index `n-2`, which is likely incorrect. It should be using `n-1`.
d. Error 4: The code calculates the maximum profit value using `opt[N][W]`, but this should be `opt[N][W]` for the actual result.

2. Effective Category: Category B (Data Declaration Errors) and Category C (Computational Errors).
3. Unidentified Error Types: The program inspection did not identify potential logical errors, such as the correctness of the knapsack algorithm's implementation.
4. Applicability: Program inspection helps catch syntax and some semantic errors but does not verify the correctness of the algorithm's implementation.

Debugging

1. Errors:
   a. Error 1: Change `n++` to `n+1` in the for loop header.
   b. Error 2: Adjust array indexing to start from `0`.
   c. Error 3: Use `n-1` instead of `n-2` for item profit and weight in option2 calculation.
   d. Error 4: Change `opt[N][W]` to `opt[N][W]` for the actual result.
2. Breakpoints Needed: At least four breakpoints are needed to address these errors.

Corrected 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];
        int[] weight = new int[N];
        // generate random instance, items 1..N
        for (int n = 0; n & lt; 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 & lt;= N; n++) {
            for (int w = 1; w & lt;= W; w++) {
                // don&#39;t take item n
                int option1 = opt[n - 1][w];
```

```
                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n - 1] & lt;= w)
                    option2 = profit[n - 1] + opt[n - 1][w - weight[n -
1]];
                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 & gt; option1);
            }
        }
        // determine which items to take
        boolean[] take = new boolean[N];
        for (int n = N, w = W; n & gt; 0; n--) {
            if (sol[n][w]) {
                take[n - 1] = true;
                w = w - weight[n - 1];
            } else {
                take[n - 1] = false;
            }
        }
        // print results
        System.out.println(& quot; item & quot; + & quot; \t & quot; + &
quot; profit & quot; + & quot; \t & quot; + & quot; weight & quot; + &
quot; \t & quot; + & quot; take & quot;);
        for (int n = 0; n & lt; N; n++) {
            System.out.println((n + 1) + & quot; \t & quot; + profit[n] +
& quot; \t & quot; + weight[n] + & quot; \t & quot; + take[n]);
        }
    }
```

Q4 Magic Number

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

Corrected Code

```
import java.util.*;
public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
        System.out.println(& quot;Enter the number to be checked.& quot;);
        int n = ob.nextInt();
        int sum = 0, num = n;
        while (num & gt; 9) {
            sum = num;
            int s = 0;
            while (sum & gt; 0) { // Changed the loop condition
                s = s * (sum % 10); // Fixed missing semicolon and updated
computation
                sum = sum / 10; // Fixed missing semicolon and updated
computation
            }
            num = s;
        }
        if (num == 1) {
            System.out.println(n + & quot; is a Magic Number.& quot;);
        } else {
            System.out.println(n + & quot; is not a Magic Number.& quot;);
        }
    }
}
```

Program Inspection

1. Errors: a. Error 1: In the inner while loop, the condition is while (sum == 0), which means the loop will only run if sum starts at 0, leading to an infinite loop. This condition should be changed. b. Error 2: There are missing semicolons at the end of the lines with sum=sum%10 and s=s*(sum/10).

2. Effective Category:
    - Category A: Data Reference Errors.
    - Category C: Computation Errors.
3. Unidentified Issues:
    - The program inspection found lexical and computation errors, but it may not catch potential logical errors, such as the loop condition and the calculations inside the loops.
4. Applicability:
    - Program inspection helps identify syntax and computation errors. To find and fix logical errors in the code, further testing and debugging are necessary.

Debugging

1. Errors: a. Error 1: Change the condition of the inner while loop from while (sum == 0) to while (sum > 0) to prevent an infinite loop. b. Error 2: Add semicolons at the end of the lines with sum=sum%10 and s=s*(sum/10) to correct the syntax errors.
2. Breakpoints Needed:
    - At least two breakpoints are needed to help fix these errors.

Q5 Merge Sort

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

Program Inspection

1. **Errors:** a. Error 1: In the mergeSort method, the function calls leftHalf(array+1) and rightHalf(array1). It should instead call leftHalf(array) and rightHalf(array). b. Error 2: The merge method is missing, even though it is called in the mergeSort method.
2. **Effective Category:**
    - Category C: Computation Errors.
3. **Unidentified Issues:**
    - The program inspection found computation errors but might not detect potential logical errors in the sorting algorithm.
4. **Applicability:**
    - Program inspection is helpful for finding syntax and computation errors. To thoroughly test and fix logical errors, additional techniques (like debugging and creating test cases) are needed.

Debugging

1. **Errors:** a. Error 1: In the mergeSort method, change int[] left = leftHalf(array+1); to int[] left = leftHalf(array); and int[] right = rightHalf(array-1); to int[] right = rightHalf(array);. b. Error 2: The code calls the merge method, but it's not included. A correct implementation of the merge method is required for the code to function properly.
2. **Breakpoints Needed:**
    - At least two breakpoints are needed to help fix these errors.

Corrected Code

```
import java.util.*;
public class MergeSort {
    public static void main(String[] args) {
        int[] list = { 14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println(& quot; before: & quot; +
Arrays.toString(list));
        mergeSort(list);
        System.out.println(& quot; after: & quot; +
Arrays.toString(list));
    }
    public static void mergeSort(int[] array) {
        if (array.length & gt; 1) {
            int[] left = leftHalf(array);
            int[] right = rightHalf(array);
            mergeSort(left);
            mergeSort(right);
            merge(array, left, right);
```

```
        }
    }
    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i & lt; size1; i++) {
            left[i] = array[i];
        }
        return left;
    }
    public static int[] rightHalf(int[] array) {
        nt size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i & lt; size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }
    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;
        for (int i = 0; i & lt; result.length; i++) {
            if (i2 & gt;= right.length || (i1 & lt; left.length & amp;&
amp; left[i1] & lt;= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

Q6 Matrix Multiplication

```
//Java program to multiply two matrices
import java.util.Scanner;
class MatrixMultiplication {
    public static void main(String args[]) {
        int m, n, p, q, sum = 0, c, d, k;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of first
matrix");
        m = in.nextInt();
```

```
        n = in.nextInt();
        int first[][] = new int[m][n];
        System.out.println("Enter the elements of first matrix");
        for (c = 0; c < m; c++)
            for (d = 0; d < n; d++)
                first[c][d] = in.nextInt();
        System.out.println("Enter the number of rows and columns of second
matrix");
        p = in.nextInt();
        q = in.nextInt();
        if (n != p)
            System.out.println("Matrices with entered orders can't be
multiplied with each other.");
        else {
            int second[][] = new int[p][q];
            int multiply[][] = new int[m][q];
            System.out.println("Enter the elements of second matrix");
            for (c = 0; c < p; c++)
                for (d = 0; d < q; d++)
                    second[c][d] = in.nextInt();
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++) {
                    for (k = 0; k < p; k++) {
                        sum = sum + first[c - 1][c - k] * second[k - 1][k
- d];
                    }
                    multiply[c][d] = sum;
                    sum = 0;
                }
            }
            System.out.println("Product of entered matrices:-");
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++)
                    System.out.print(multiply[c][d] + "\t");
                System.out.print("\n");
            }
        }
    }
}
```

Program Inspection

1. **Errors:** a. Error 1: In the nested loop for calculating the product of matrices, change sum = sum + first[c-1][c-k]*second[k-1][k-d]; to sum = sum + first[c][k] * second[k][d];. The indices should start from 0. b. Error 2: The variables c, d, and k need to be properly initialized within the for loops.
2. **Effective Categories:**

- Category C: Computation Errors.
- Category B: Data Declaration Errors.

3. **Unidentified Issues:**
   - The program inspection identified computation errors but may not catch potential logical errors in the matrix multiplication algorithm.
4. **Applicability:**
   - Program inspection is useful for detecting syntax and computation errors. For thorough testing and fixing logical errors, additional techniques (like debugging and creating test cases) are necessary.

Debugging

1. **Errors:** a. Error 1: In the nested loop for matrix multiplication, change sum = sum + first[c-1][c-k]*second[k-1][k-d]; to sum = sum + first[c][k] * second[k][d];. b. Error 2: Properly initialize the variables c, d, and k within the for loops.
2. **Breakpoints Needed:**
   - At least two breakpoints are needed to help address these errors.

Corrected Code

```
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(& quot;Enter the number of rows and columns of
the first matrix & quot;);
        m = in.nextInt();
        n = in.nextInt();
        int first[][] = new int[m][n];
        System.out.println(& quot;Enter the elements of the first matrix &
quot;);
        for (c = 0; c & lt; m; c++)
            for (d = 0; d & lt; n; d++)
                first[c][d] = in.nextInt();
        System.out.println(& quot;Enter the number of rows and columns of
the second matrix & quot;);
        p = in.nextInt();
        q = in.nextInt();
        if (n != p)
            System.out.println(& quot;Matrices with entered orders can
&#39;t be multiplied with each other.& quot;);
        else {
```

```
            int second[][] = new int[p][q];
            int multiply[][] = new int[m][q];
            System.out.println(& quot;Enter the elements of the second
matrix & quot;);
            for (c = 0; c & lt; p; c++)
            for (d = 0; d & lt; q; d++)
            second[c][d] = in.nextInt();
            for (c = 0; c & lt; m; c++) {
                for (d = 0; d & lt; q; d++) {
                    sum = 0;
                    for (k = 0; k & lt; p; k++) {
                        sum = sum + first[c][k] * second[k][d];
                    }
                    multiply[c][d] = sum;
                }
            }
            System.out.println(& quot;Product of entered matrices: -&
quot;);
            for (c = 0; c & lt; m; c++) {
                for (d = 0; d & lt; q; d++)
                System.out.print(multiply[c][d] + & quot; \t & quot;);
                System.out.println();
            }
        }
    }
}
```

Q7 Quadratic Probing

```
/**
* 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');
    }
}
```

Program Inspection

1. **Errors:** a. Error 1: In the insert method, there is a logical mistake in the line i + = (i + h / h--) % maxSize;. The correct line should be i = (i + h * h++) % maxSize; to implement quadratic probing. b. Error 2: In the get method, the loop condition should account for a full table. Change while (keys[i] != null) to for (int j = 0; j < maxSize; j++) to ensure the loop stops eventually. c. Error 3: In the remove method, the loop for rehashing keys needs adjustment. Change for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) to for (int j = 0; j < maxSize; j++) to ensure proper rehashing.

2. **Effective Categories:**
    - Category C: Computation Errors.
    - Category E: Control-Flow Errors.
3. **Unidentified Issues:**
    - The program inspection found computation errors but might not catch potential logical errors in the hash table operations.
4. **Applicability:**
    - Program inspection is helpful for identifying syntax and computation errors. For thorough testing and fixing logical errors, additional techniques (like debugging and creating test cases) are needed.

Debugging

1. **Errors:** a. Error 1: In the insert method, change i + = (i + h / h--) % maxSize; to i = (i + h * h++) % maxSize;. b. Error 2: In the get method, update the loop condition to for (int j = 0; j < maxSize; j++) to prevent it from running indefinitely. c. Error 3: In the remove method, change the rehash loop to for (int j = 0; j < maxSize; j++) to ensure proper rehashing.
2. **Breakpoints Needed:**
    - At least three breakpoints are necessary to address these errors.

Corrected Code

```
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 the hash table **/
```

```
    public int getSize() {
        return currentSize;
    }
    /** Function to check if the hash table is full **/
    public boolean isFull() {
        return currentSize == maxSize;
    }
    /** Function to check if the hash table is empty **/
    public boolean isEmpty() {
        return getSize() == 0;
    }
    /** Function to check if the hash table contains a key **/
    public boolean contains(String key) {
        return get(key) != null;
    }
    /** Function to get hash code of a given key **/
    private int hash(String key) {
        return key.hashCode() % maxSize;
    }
    /** 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;
        for (int j = 0; j & lt; maxSize; j++) {
            if (keys[i] != null) {
                if (keys[i].equals(key)) {
                    return vals[i];
                }
                i = (i + h * h++) % maxSize;
                System.out.println(& quot; i & quot; + i);
            } else {
```

```
                return null;
            }
        }
        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 (int j = 0; j & lt; maxSize; j++) {
            if (keys[i] != null) {
                String tmp1 = keys[i], tmp2 = vals[i];
                keys[i] = vals[i] = null;
                currentSize--;
                insert(tmp1, tmp2);
            } else {
                currentSize--;
                break;
            }
        }
    }
    /** Function to print HashTable **/
    public void printHashTable() {
        System.out.println(& quot; \nHash Table: & quot;);
        for (int i = 0; i & lt; maxSize; i++) {
            if (keys[i] != null) {
                System.out.println(keys[i] + & quot; & quot; + 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(& quot;Hash Table Test\n\n & quot;);
        System.out.println(& quot;Enter size & quot;);
        /** Make an object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());
```

```
        char ch;
        /** Perform QuadraticProbingHashTable operations **/
        do {
            System.out.println(& quot; \nHash Table Operations\n & quot;);
            System.out.println(& quot; 1. insert & quot;);
            System.out.println(& quot; 2. remove & quot;);
            System.out.println(& quot; 3. get & quot;);
            System.out.println(& quot; 4. clear & quot;);
            System.out.println(& quot; 5. size & quot;);
            int choice = scan.nextInt();
            switch (choice) {
                case 1:
                    System.out.println(& quot;Enter key and value &
quot;);
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2:
                    System.out.println(& quot;Enter key & quot;);
                    qpht.remove(scan.next());
                    break;
                case 3:
                    System.out.println(& quot;Enter key & quot;);
                    System.out.println(& quot; Value = & quot; +
qpht.get(scan.next()));
                    break;
                case 4:
                    qpht.makeEmpty();
                    System.out.println(& quot;Hash Table Cleared\n &
quot;);
                    break;
                case 5:
                    System.out.println(& quot; Size = & quot; +
qpht.getSize());
                    break;
                default:
                    System.out.println(& quot;Wrong Entry \n & quot;);
                    break;
            }
            /** Display hash table **/
            qpht.printHashTable();
            System.out.println(& quot; \nDo you want to continue (Type y
or n) \n & quot;);
            ch = scan.next().charAt(0);
        } while (ch == &#39; Y &#39; || ch == &#39; y &#39;);
    }
}
```

Q8 Sorting Array

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

Program Inspection

1. **Errors:** a. Error 1: In the first for loop, the condition for (int i = 0; i <= n; i++); is incorrect. It uses i <= n, which can lead to the loop body not executing. b. Error 2: The condition in the second for loop should also be corrected; it currently has the same issue as the first.
2. **Effective Categories:**
    - Category C: Computation Errors.

- Category D: Comparison Errors.

3. **Unidentified Issues:**
    - The program inspection identified computation errors but may not catch potential logical errors in the sorting logic.
4. **Applicability:**
    - Program inspection is helpful for detecting syntax and computation errors. For thorough testing and fixing logical errors, additional techniques (like debugging and creating test cases) are necessary.

Debugging

1. **Errors:** a. Error 1: Change the first for loop from for (int i = 0; i <= n; i++); to for (int i = 0; i < n; i++) to iterate through the array correctly. b. Error 2: Change the second for loop from for (int i = 0; i <= n; i++) to for (int i = 0; i < n; i++) to iterate through the array correctly.
2. **Breakpoints Needed:**
    - Two breakpoints are necessary to address these errors.
3. **Steps Taken to Fix Errors:** a. In the first for loop, change the condition to i < n to iterate through the array properly. b. In the second for loop, change the condition to i < n to iterate through the array correctly.

Corrected Code

```
import java.util.Scanner;
public class Ascending_Order {
    public static void main(String[] args) {
    int n, temp;
    Scanner s = new Scanner(System.in);
    System.out.print(& quot;Enter no.of elements you want in the array:&
quot;);
    n = s.nextInt();
    int a[] = new int[n];
        System.out.println(& quot;Enter all the elements:& quot;);
        for (int i = 0; i & lt; n; i++) {
            a[i] = s.nextInt();
        }
        for (int i = 0; i & lt; n; i++) {
            for (int j = i + 1; j & lt; n; j++) {
                if (a[i] & gt; a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
```

```
            }
        }
        System.out.print(& quot;Ascending Order:& quot;);
        for (int i = 0; i & lt; n - 1; i++) {
            System.out.print(a[i] + & quot;,& quot;);
        }
        System.out.print(a[n - 1]);
    }
}
```

Q9 Stack Implementation

```
//Stack implementation in java
import java.util.Arrays;
public class StackMethods {
    private int top;
    int size;
    int[] stack;
    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;
    }
    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack is full, can't push a value");
        }
        else {
            top--;
            stack[top] = value;
        }
    }
    public void pop() {
        if (!isEmpty())
            top++;
        else {
            System.out.println("Can't pop...stack is empty");
        }
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public void display() {
        for (int i = 0; i > top; i++) {
            System.out.print(stack[i] + " ");
        }
```

```
        System.out.println();
    }
}
public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);
        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

Program Inspection

1. **Errors:** a. Error 1: In the push method, the top variable should be incremented before adding a value to the stack. Currently, it is being decremented, which causes incorrect behavior. b. Error 2: In the display method, the loop condition uses > instead of <, which prevents the stack from displaying correctly.
2. **Effective Categories:**
    - Category C: Computation Errors.
    - Category D: Comparison Errors.
3. **Unidentified Issues:**
    - The inspection found computation errors, but it may not catch potential logical errors in stack behavior, such as handling overflow or underflow.
4. **Applicability:**
    - Program inspection is useful for identifying syntax and computation errors. However, for more thorough testing, additional techniques are necessary.

Debugging

1. **Errors:** a. Error 1: In the push method, change top-- to top++ to properly increment the top before adding a value. b. Error 2: In the display method, update the loop condition from for(int i=0; i > top; i++) to for (int i = 0; i < top; i++) to correctly display the stack.

2. **Breakpoints Needed:**
    - Two breakpoints are required to address these errors.
3. **Steps Taken to Fix Errors:** a. In the push method, replace top-- with top++. b. In the display method, adjust the loop condition to use < instead of >.

Corrected Code

```
import java.util.Arrays;
public class StackMethods {
    private int top;
    int size;
    int[] stack;
    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;
    }
    public void push(int value) {
        if (top == size - 1) {
            System.out.println(& quot;Stack is full, can &#39;t push a
value & quot;);
        } else {
            top++;
            stack[top] = value;
        }
    }
    public void pop() {
        if (!isEmpty())
            top--;
        else {
            System.out.println(& quot; Can &#39;t pop...stack is empty &
quot;);
        }
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public void display() {
        for (int i = 0; i & lt; top; i++) {
            System.out.print(stack[i] + & quot; & quot;);
        }
        System.out.println();
    }
}
public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
```

```
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);
        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

Q10 Tower Of Hanoi

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

Program Inspection

1. **Errors:** a. Error 1: There is incorrect use of post-increment and post-decrement operators (topN++, inter--). b. Error 2: The parameters in the recursive call to doTowers are in the wrong order.
2. **Effective Category:**
    - Category C: Computation Errors.
3. **Unidentified Issues:**
    - The inspection may not catch more complex logic errors or algorithmic problems.
4. **Applicability:**
    - The program inspection technique is helpful for finding syntax and logical errors, but it should be used alongside other testing methods.

Debugging

1. **Errors:** a. Error 1: Correct the use of post-increment and post-decrement operators by changing topN++ to topN--. b. Error 2: Change the order of parameters in the recursive call to doTowers.
2. **Breakpoints Needed:**
    - At least two breakpoints are required to fix these errors.
3. **Steps Taken to Fix Errors:** a. Replace topN++ with topN--. b. Swap the order of from and inter in the recursive call.

Corrected Code

```
public class MainClass {
    public static void main(String[] args) {
    int nDisks = 3;
        doTowers(nDisks, &#39; A &#39;, &#39; B &#39;, &#39; C &#39;);
    }
    public static void doTowers(int topN, char from, char inter, char to)
{
        if (topN == 1) {
            System.out.println(& quot;Disk 1 from & quot; + from + & quot;
to & quot; + to);
        } else {
            doTowers(topN - 1, from, to, inter);
            System.out.println(& quot; Disk & quot; + topN + & quot; from
& quot; + from + & quot; to
                & quot; + to);
            doTowers(topN - 1, inter, from, to);
        }
    }
```

```
}
```