# IT314 Software Engineering
# Lab 7
# By: 202201446

---

# Part 1

## Armstrong

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

- The class Armstrong misses a closing curly bracket '}'.
- The computation for determining the Armstrong number is wrong

Q.2) Which category of program inspection would you find more effective?
Ans) In this program, the category C(Computation Errors) is effective.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) Syntax Error

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes

## GCD and LCM

Q.1) How many errors are there in the program? Mention the errors you have identified.
Ans) Only one error in this program, which is mentioned in the comment of the program.

Q.2) Which category of program inspection would you find more effective?
Ans) The Comparison Error category D.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) None

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes

# Knapsack

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

- In the line int option1 = opt[n++][w];, n++ increments n prematurely. It should be opt[n][w] instead.
- The condition if (weight[n] > w) should be if (weight[n] <= w) to check if the weight can be included in the knapsack.
- In the line option2 = profit[n-2] + opt[n-1][w-weight[n]];, profit[n-2] should be profit[n] since we are considering item n.
- The logic for updating the sol array depends on incorrect computation results from option1 and option2.

Q.2) Which category of program inspection would you find more effective?
Ans) In this program, Category C (Computation Errors) would be the most effective since the main issue is in the calculation logic of the knapsack problem.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) Syntax Error is not easily identified through program inspection since the syntax appears correct. However, logical errors within the syntax (such as misuse of operators like n++) require deeper inspection or testing.

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes, program inspection is helpful, especially in identifying logical errors like those in computation. It provides a structured way to catch errors before runtime debugging.

# Magic Number

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

- In the inner while loop, the condition while (sum == 0) should be while(sum > 0) to iterate correctly through the digits of sum.
- The line s = s * (sum / 10); should be s = s + (sum % 10); to correctly calculate the sum of the digits.
- The line sum = sum % 10 is missing a semicolon at the end.
- The logic for the outer while(num > 9) is flawed, as it should be summing the digits rather than assigning sum = num at the beginning of each iteration. The initialization of the sum inside the loop is unnecessary.

Q.2) Which category of program inspection would you find more effective?
Ans) In this program, focusing on Computation Errors would be more effective since the errors primarily affect the logic of digit summation.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) Syntax Error is harder to spot through inspection alone, as it involves missing semicolons and would typically be caught during compilation.

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes, program inspection is useful for identifying logical flaws and incorrect conditions in loops, even though some syntax errors might be missed without compilation.

## Merge Sort

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

- In the mergeSort method, int[] left = leftHalf(array + 1); should be int[] left = leftHalf(array); to pass the full array instead of incorrectly modifying it with +1.
- Similarly, int[] right = rightHalf(array - 1); should be int[] right = rightHalf(array); as subtracting 1 from the array is invalid.
- In the merge call, merge(array, left++, right--); should be merge(array, left, right); since left++ and right-- incorrectly modify the references to the arrays.

Q.2) Which category of program inspection would you find more effective?
Ans) Computation Errors would be more effective in this program, as the main issues lie in how the arrays are split and passed during the recursive merge sort process.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) Syntax Errors such as missing semicolons or minor typographical errors would not be easily identified during inspection unless the program is compiled.

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes, program inspection is useful in identifying logical and computational mistakes, especially in recursive algorithms like merge sort.

## Multiply Matrices

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

- In the multiplication logic, first[c-1][c-k] should be first[c][k] to correctly reference the element from the first matrix. The indices c-1 and c-k are invalid and result in incorrect array indexing.
- Similarly, second[k-1][k-d] should be second[k][d] to properly reference the element from the second matrix.
- The program prompts "Enter the number of rows and columns of first matrix" twice. The second prompt should ask for the second matrix instead.

Q.2) Which category of program inspection would you find more effective?
Ans) Computation Errors would be the most effective for this program, as the key errors lie in how the indices for matrix multiplication are computed.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) Syntax Errors such as incorrect array index or small typos like missing semicolons would be harder to identify through manual inspection and would be better caught through compilation.

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes, it helps identify logical issues in matrix operations, especially in loops where index management is crucial for correct results.

# Quadratic Probing

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

- In the insert method, there is an incorrect syntax with the line i + = (i + h / h--) % maxSize;. It should be i = (i + h * h++) % maxSize;.
- In the remove method, the currentSize is decremented twice, which leads to an incorrect size count. One of the decrements should be removed.
- The comment /** maxSizeake object of QuadraticProbingHashTable **/ contains a typo and should be corrected to /** Make object of QuadraticProbingHashTable **/.

Q.2) Which category of program inspection would you find more effective?
Ans) Computation Errors would be more effective in this program, as the main issues lie in how the hash table operations are implemented, particularly in the insert and remove methods where index calculations and size management are crucial.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans)  Runtime errors or edge cases that might cause issues with very large hash tables or specific input patterns would be difficult to identify through inspection alone. These might include potential integer overflow in the hash function or performance degradation with certain key distributions.

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes, program inspection is worth applying to this code. It helps identify logical errors, computational mistakes, and inconsistencies in the implementation of complex operations like quadratic probing and rehashing, which might be overlooked during casual reading or even some forms of testing.

# Sorting Array

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

- The class name "Ascending _Order" contains a space, which is invalid in Java. It should be "AscendingOrder".
- In the outer sorting loop, for (int i = 0; i >= n; i++); is incorrect. It should be for (int i = 0; i < n; i++) without the semicolon at the end.
- The condition a[i] <= a[j] in the inner loop will result in descending order, not ascending. It should be a[i] > a[j] for ascending order.

Q.2) Which category of program inspection would you find more effective?
Ans) Control Flow Errors would be more effective in this program, as the main issues lie in the loop conditions and logic that control the sorting process.

Q.3) Which type of error you are not able to identified using the program inspection?
Ans) Runtime performance issues, such as the inefficiency of the bubble sort algorithm for large arrays, would not be easily identified through code inspection alone. These would require runtime analysis and performance testing.

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes, program inspection is valuable for this code. It helps identify logical errors in the sorting algorithm, syntax issues in class naming, and mistakes in loop conditions. These errors significantly affect the program's correctness and would be challenging to debug solely through testing.

# Stack Implementation

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

- The push method decreases the top index before assigning the value, which leads to an incorrect stack behavior.
- The display method's loop condition i > top is incorrect, so it does not display the stack properly.

Q.2) Which category of program inspection would you find more effective?
Ans)  In this program, the computation errors are effective.

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

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes

# Tower of Hanoi

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

- The doTowers method has an infinite loop caused by the use of topN ++, which should be topN - 1 to correctly decrement the disk count for the recursive call.
- The parameters inter-- and from+1 in the recursive call are incorrect, as they are trying to modify the character values, which is not valid in this context.

Q.2) Which category of program inspection would you find more effective?
Ans) In this program, the computation errors are effective.

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

Q.4) Is the program inspection technique is worth applicable?
Ans) Yes