IT-314 Software Engineering

Lab7: Debugging and Inspection

ID: 202201250

PARTH PRAJAPATI

Program Inspection

❖ Armstrong Number

Errors Identified:

- There are two errors in the program:
  - Incorrect calculation of the remainder: remainder = num / 10; should be remainder = num % 10;.
  - Incorrect update of num: num = num % 10; should be num = num / 10;.

Effective Category:

- The most effective category of program inspection for this code is **Category C: Computation Errors**, as the errors pertain to arithmetic operations and computation logic.

Unidentified Errors:

- Program inspection does not identify runtime errors such as handling non-numeric input or other exceptions. It also does not cover style and maintainability issues comprehensively.

Applicability:

- The program inspection technique is valuable for identifying and rectifying issues related to code structure and computation errors, ensuring that the program logic is correct before running the code.

Breakpoints Used:

1. One breakpoint before the while loop to check the initial value of num.
2. Another breakpoint inside the while loop to monitor the values of remainder and check.

Steps Taken to Fix the Error:

- Corrected the operations within the while loop by swapping the use of / and %.

❖ GCD and LCM of two given numbers

Errors Identified:

- **GCD Calculation:**
  - The condition in the while loop should be

    while(a % b != 0).

    Current condition while(a % b == 0) is incorrect.
- **LCM Calculation:**
  - The condition in the if statement should be

    if(a % x == 0 && a % y == 0).

    Current condition if(a % x != 0 && a % y != 0) is incorrect.
  - In the LCM calculation, it should break the loop and return a when both x and y are factors of a.

Effective Category:

- The most effective category of program inspection for this code is **Category C: Computation Errors**, as the errors pertain to logical and arithmetic operations.

Unidentified Errors:

- Program inspection may not identify runtime errors such as invalid input (non-integer values). It also does not address the efficiency and optimization of the code.

Applicability:

- The program inspection technique is valuable for identifying and rectifying logical and computation errors, ensuring the correct functionality of the algorithm.

Breakpoints Used:

1. One breakpoint before the while loop in the gcd method to check the initial values of a and b.
2. Another breakpoint inside the while loop to monitor the remainder calculation.
3. A breakpoint inside the lcm loop to track the increments of a and check divisibility.

Steps Taken to Fix the Error:

- Corrected the while condition in the gcd function.
- Fixed the condition in the lcm function to check if both numbers divide a evenly.

❖ Knapsack

Errors Identified:

- **Loop Increment Error:**
  - In the option1 calculation, opt[n++][w] should be opt[n-1][w].
- **Index Error:**
  - In the option2 calculation, profit[n-2] should be profit[n].

Effective Category:

- The most effective category of program inspection for this code is **Category C: Computation Errors**, as the errors pertain to array indexing and logical errors in the computation.

Unidentified Errors:

- Program inspection may not identify runtime errors such as invalid input or edge cases where the maximum weight W is zero. It also does not address the efficiency and performance of the algorithm.

Applicability:

- The program inspection technique is valuable for identifying and rectifying logical and computation errors, ensuring the correct functionality of the algorithm.

Breakpoints Used:

1. One breakpoint before the for loop to check the initialized values of profit and weight.

2. A breakpoint inside the nested loops to monitor the values of opt[n][w] and sol[n][w] during the calculation of option1 and option2.
3. A breakpoint during the item selection process to ensure the correct items are being chosen.

Steps Taken to Fix the Error:

- Corrected the usage of the increment operator in the line int option1 = opt[n][w]; to avoid skipping indices.
- Adjusted the conditional check if (weight[n] <= w) to ensure the item is selected if it fits within the knapsack.
- Fixed the array index issue in the profit and opt matrix calculations.

❖ Magic number

Errors Identified:

- **Inner Loop Condition Error:**
  - while(sum==0) should be while(sum!=0).
- **Multiplication Error:**
  - s=s*(sum/10) should be s=s+(sum%10).
- **Semicolon Missing:**
  - sum=sum%10 should be sum=sum/10;.

Effective Category:

- The most effective category of program inspection for this code is **Category C: Computation Errors**, as the errors pertain to logical and arithmetic operations within the loops.

Unidentified Errors:

- Program inspection may not identify runtime errors such as input validation and edge cases where the input number is zero or negative.

Applicability:

- The program inspection technique is valuable for identifying and rectifying logical and computation errors, ensuring the correct functionality of the algorithm.

Breakpoints Used:

1. Before the while(num > 9) loop to check the initial value of num.

2. Inside the second while loop to track the process of summing the digits.

Steps Taken to Fix the Error:

- Changed the loop condition to while(sum != 0).
- Corrected the expression to add digits properly using s = s + (sum % 10).

❖ Merge sort algorithm

Errors Identified:

- **Array Modification in mergeSort Function:**
  - int[] left = leftHalf(array+1); should be int[] left = leftHalf(array);.
  - int[] right = rightHalf(array-1); should be int[] right = rightHalf(array);.
- **Index Modification in merge Function:**
  - merge(array, left++, right--); should be merge(array, left, right);.

Effective Category:

- The most effective category of program inspection for this code is **Category C: Computation Errors**, as the errors pertain to incorrect modifications and accesses of array elements.

Unidentified Errors:

- Program inspection may not identify issues related to performance optimizations and edge cases for very large arrays where recursion depth might be a concern.

Applicability:

- The program inspection technique is valuable for identifying and rectifying logical and computation errors, ensuring the correct implementation of the merge sort algorithm.

Breakpoints Used:

1. A breakpoint before the recursive mergeSort() call to monitor the splitting of arrays.

2. A breakpoint inside the merge() function to track how elements from left and right arrays are merged.

Steps Taken to Fix the Errors:

- Removed incorrect arithmetic from leftHalf(array+1) and rightHalf(array-1) and passed the correct array values.
- Fixed the merging step by avoiding unnecessary increments and decrements when calling merge().

❖ Multiplication two matrices

Errors Identified:

- **Indexing Error in Matrix Multiplication Logic:**
    - The statements first[c-1][c-k] and second[k-1][k-d] are incorrect; they should be first[c][k] and second[k][d] respectively to access the correct elements for multiplication.

  Resetting the Sum:

    - sum is reset correctly after each multiplication but should be initialized properly at the start of the multiplication loop for clarity.
- **Input Prompts Error:**
    - The prompt for the second matrix should say "Enter the number of rows and columns of the second matrix" instead of repeating "first matrix".

Effective Category:

- The most effective category of program inspection for this code is **Category C: Computation Errors**, as the identified issues pertain to incorrect mathematical operations during matrix multiplication.

Unidentified Errors:

- Program inspection may not identify issues related to performance inefficiencies, such as the algorithm's time complexity when dealing with larger matrices.

Applicability:

- The program inspection technique is valuable for identifying logical errors in matrix operations and ensuring that the implementation adheres to mathematical principles.

Breakpoints Used:

1. A breakpoint at the nested loop inside the multiplication process to track how the values are being accessed from both matrices.

Steps Taken to Fix the Errors:

- Corrected the matrix element access by changing first[c-1][c-k] to first[c][k] and second[k-1][k-d] to second[k][d].

❖ Quadratic Probing Hash Table

Errors Identified:

- **Syntax Error in Incrementing i:**
  - The line i + = (i + h / h--) % maxSize; has an extra space which causes a compilation error. It should be i += (i + h * h) % maxSize;.
- **Hash Calculation Logic Error in get Method:**
  - The expression i = (i + h * h++) % maxSize; incorrectly uses h++. It should be h = h + 1; after using h * h to ensure the correct increment of h for the next iteration.
- **Misleading Comments:**
  - The comments for insert, get, and remove functions use the wrong spelling and may confuse readers.
- **Incorrect Resizing Logic:**
  - The makeEmpty function creates new arrays for keys and vals, which may not clear the existing data correctly as references remain. It should reset the existing arrays instead of creating new ones.

Effective Category:

- The most effective category of program inspection for this code is **Category B: Data Structure Errors**, as the errors primarily relate to the logic of maintaining the hash table and the quadratic probing mechanism.

Unidentified Errors:

- Program inspection may not identify potential performance issues, such as the hash table's behavior under heavy load, which could lead to performance degradation due to increased collision handling.

Applicability:

- The program inspection technique is indeed applicable and beneficial, as it helps identify logical errors and ensures that the hash table operations follow the expected behavior.

Breakpoints Used:

1. A breakpoint at the insertion logic to ensure that keys are being placed correctly in the hash table.
2. A breakpoint in the retrieval logic to check if keys can be fetched properly.

Steps Taken to Fix the Errors:

- Corrected the increment logic in the insertion, retrieval, and removal processes.
- Ensured that hash values are always positive by using Math.abs().
- Revised the comments for better clarity and correctness.

- Sorting the array in ascending order

Errors Identified:

1. **Syntax Error in Incrementing i:**
    - **Correction:** Change i + = (i + h / h--) % maxSize; to i += (i + h * h) % maxSize;.
2. **Hash Calculation Logic Error in get Method:**
    - **Correction:** Replace i = (i + h * h++) % maxSize; with h = h + 1; after using h * h.
3. **Misleading Comments:**
    - **Correction:** Revise comments for clarity and correct spelling.
4. **Incorrect Resizing Logic in makeEmpty:**
    - **Correction:** Reset existing arrays instead of creating new ones.

Effective Category:

- **Category B: Data Structure Errors** due to issues in hash table logic.

Unidentified Errors:

- **Potential Performance Issues:** Not addressing behavior under high load could lead to degraded performance.

Applicability:

- Program inspection is beneficial for identifying logical errors and ensuring expected hash table operations.

Breakpoints Used:

1. A breakpoint was placed in the sorting loop to observe how elements are swapped.
2. A breakpoint was placed before printing the sorted array to verify the final array content.

Steps Taken to Fix the Errors:

- Corrected the class name and loop conditions.
- Removed the unnecessary semicolon.
- Fixed the comparison logic in the sorting condition.

❖ Stack implementation

Errors Identified:

1. **Logic Error in push Method:**
    - Issue: The top index is decremented before storing the value.
    - Correction: Change top-- to top++ to push values correctly.
2. **Logic Error in pop Method:**
    - Issue: The top index is incremented correctly but does not remove the value from the stack.
    - Correction: You should handle the return value and reset the stack at top.
3. **Logic Error in display Method:**
    - Issue: The loop condition is incorrect (i > top should be i <= top).
    - Correction: Change for(int i=0;i>top;i++) to for(int i=0; i <= top; i++).

Effective Category:

- **Category B: Data Structure Errors**
    - The errors relate to the logic of stack operations and index management.

Unidentified Errors:

- Potential array index out-of-bounds errors during push and pop operations under high usage may not be identified.

Applicability:

- The program inspection technique is applicable as it helps identify logical errors in stack operations, ensuring proper functionality.

❖ Tower of Hanoi

Errors Identified:

1. **Logic Error in the Recursive Calls:**
    - Issue: The parameters in the recursive call doTowers(topN ++, inter--, from+1, to+1) incorrectly modify topN and inter. This will not work as intended and cause incorrect behavior.
    - Correction: Use topN - 1 and inter directly without modifying them.
2. **Output Logic Error:**
    - The logic to print "Disk 1 from A to C" is correct for the base case, but subsequent recursive calls are not correctly handling the parameters.
    - Correction: Ensure the parameters are passed correctly to maintain the correct state of the disks.
3. **Unnecessary Parameter Modifications:**
    - Incrementing and decrementing topN and inter in the recursive call is incorrect.
    - Correction: Use the original values without modifying them in the call.

Effective Category:

- **Category A: Logic Errors**
    - The errors primarily relate to the logic of recursive function calls and parameter handling.

Unidentified Errors:

- Potential stack overflow errors if the number of disks (nDisks) is too large may not be identified during a basic inspection.

Applicability:

- The program inspection technique is applicable, as it helps identify logical errors in the recursive implementation of the Tower of Hanoi algorithm.