Software Engineering(IT-314)

Lab Session VII

Program Inspection, Debugging and Static Analysis

Submitted By : Kkavy Dave (202201421)

Question 1 : Program Inspection

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?

Answer :

Errors found and related to their category

GitHubLink:https://github.com/x64dbg/x64dbg/blob/development/src/bridge/bridgemain.cpp

Category A: Data Reference Errors

- **Line 128**: The pointer backslash is checked for null, but there is no check on szUserDirectory size before usage.
- **Line 300**: In *BridgeLoadLibraryCheckedW*, the *szDll* parameter could potentially be null if not handled carefully.

Category B: Data-Declaration Errors

- **Line 12**: *HINSTANCE hInst* is declared globally but could lead to issues if modified elsewhere. It would be better to encapsulate its scope.
- **Line 228**: *userDirUtf8* vector resizing could lead to errors if not properly checked for size bounds after reading the file.

Category C: Computation Errors

- **Line 534**: The *sscanf_s* method is used to convert values, but there's no check on the return value. This could cause an issue if the input string isn't correctly formatted.

Category D: Comparison Errors

- **Line 1270**: Boolean conditions are mixed with comparison operators, specifically for the statement checking for the validity of text or addr. It's easy to mistakenly mix up conditions if addr is null or text is too large.

Category E: Control-Flow Errors

- **Line 475**: There is a possibility of skipping over the *loadIfExists*() function without proper error handling if DLLs aren't loaded. This should be handled with more detailed conditions to avoid unexpected control flow issues.

Category F: Interface Errors

- **Line 576**: In *BridgeSettingRead*(), the *CreateFileW* function is used, but error handling could be improved in case of permission issues or file locks.

Category G: Input/Output Errors

- **Line 528**: File reading logic does not handle cases where the file could be corrupted or partially written, causing the vector resize operation to fail unexpectedly.

Category H: Other Checks

- **Line 802**: The *GlobalAlloc*() function in *BridgeAlloc*() could fail due to low memory, but the error message handling and process exit seem a bit aggressive. A more graceful fallback or logging mechanism might be better.

Answers to the questions:

1. **How many errors are there in the program? Mention the errors you have identified.**
    - Identified potential errors across several categories: improper memory reference checks, undeclared variables, missing file permission checks, incorrect type casting, and unchecked computation results. Around 8-10 specific issues have been flagged.
2. **Which category of program inspection would you find more effective?**
    - Category A (Data Reference Errors) and Category F (Interface Errors) were particularly effective for finding issues related to memory management and function calls that lacked proper safety checks.
3. **Which type of error are you not able to identify using program inspection?**
    - Some logical errors that depend on specific runtime behavior, such as threading or race conditions, may not be apparent through static inspection alone. Category E (Control-Flow Errors) might also miss deeper, context-specific bugs during program execution.
4. **Is the program inspection technique worth applying?**
    - Yes, the inspection technique is highly valuable for catching common errors before runtime, particularly in critical systems such as this one where memory and pointer management are frequent.

Question 2 :

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?

1. Armstrong Number

Errors:

1. Syntax error in how remainder is calculated: remainder = num / 10 should be remainder = num % 10.
2. Logical error in how num is reduced: num = num % 10 should be num = num / 10.

Breakpoints Needed:

1. Before the loop to check the value of num.
2. After the loop to check the check value.

Steps to Fix:

- Corrected the remainder calculation.
- Changed how num is reduced.

Corrected Code:

```
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; // Corrected remainder calculation
                        check = check + (int)Math.pow(remainder, 3);
                        num = num / 10; // Corrected how num is reduced
                }
                if(check == n)
                        System.out.println(n + " is an Armstrong Number");
                else
                        System.out.println(n + " is not an Armstrong Number");
        }
}
```

2. GCD and LCM Program

Errors:

1. Logic error in GCD calculation: while(a % b == 0) should be while(a % b != 0).
2. Logical error in LCM: if(a % x != 0 && a % y != 0) should be if(a % x == 0 && a % y == 0).

Breakpoints Needed:

1. Before the while loop in the GCD calculation.
2. Before the LCM loop.

Steps to Fix:

- Fixed the condition in the GCD loop.
- Corrected the condition in the LCM calculation.

Corrected Code:

```
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;
    b = (x < y) ? x : y;
    while (a % b != 0) { // Fixed condition
      r = a % b;
      a = b;
      b = r;
    }
    return b;
  }
  static int lcm(int x, int y) {
    int a = (x > y) ? x : y;
    while (true) {
      if (a % x == 0 && a % y == 0) // Fixed condition
        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();
  }
}
```

3. Knapsack Problem

Errors:

1. Logic error: int option1 = opt[n++][w]; should be int option1 = opt[n-1][w];.
2. Incorrect weight and profit handling.

Breakpoints Needed:

1. Before the loop to check profit and weight assignments.
2. Before selecting the optimal solution.

Steps to Fix:

- Fixed the increment logic in the option1 assignment.
- Corrected the weight comparison logic.

Corrected Code:

```
public class Knapsack {
  public static void main(String[] args) {
    int N = Integer.parseInt(args[0]);
    int W = Integer.parseInt(args[1]);
    int[] profit = new int[N+1];
    int[] weight = new int[N+1];
    for (int n = 1; n <= N; n++) {
      profit[n] = (int) (Math.random() * 1000);
      weight[n] = (int) (Math.random() * W);
    }
    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++) {
        int option1 = opt[n-1][w]; // Fixed increment issue
        int option2 = Integer.MIN_VALUE;
        if (weight[n] <= w)
          option2 = profit[n] + opt[n-1][w - weight[n]];
        opt[n][w] = Math.max(option1, option2);
        sol[n][w] = (option2 > option1);
      }
    }
    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;
      }
    }
    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]);
        }
    }
}
```

4. Magic Number Check

Errors:

1. Incorrect condition while(sum == 0) should be while(sum != 0).
2. Missing semicolon in sum=sum%10;.

Breakpoints Needed:

1. Before the nested while to check sum accumulation.

Steps to Fix:

- Fixed the incorrect condition in the inner loop.
- Added the missing semicolon.

Corrected Code:

```
import java.util.Scanner;
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) { // Fixed condition
                s = s + (sum % 10); // Added semicolon
                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.");
        }
    }
}
```

5. Merge Sort Algorithm

Errors:

1. **Logic Error**: Array splitting is done incorrectly in leftHalf() and rightHalf() methods. The code adds/subtracts 1 unnecessarily.
2. **Incorrect Parameters**: The merge method uses left++ and right-- which is wrong.

Breakpoints Needed:

1. Before calling leftHalf() and rightHalf() to verify array splitting.
2. Before the merge() function to verify the merging process.

Steps to Fix:

- Corrected the array splitting logic.
- Fixed parameters in the merge() method.

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("before: " + Arrays.toString(list));
    mergeSort(list);
    System.out.println("after:  " + Arrays.toString(list));
  }
  public static void mergeSort(int[] array) {
    if (array.length > 1) {
      int[] left = leftHalf(array); // Removed +1
      int[] right = rightHalf(array); // Removed -1
      mergeSort(left);
      mergeSort(right);
      merge(array, left, right); // Removed ++/-- and corrected
    }
  }
  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;
  }
  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;
    }
public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;
        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

6. Matrix Multiplication

Errors:

1. Logic error in matrix element multiplication: sum = sum + first[c-1][c-k]*second[k-1][k-d]; should be sum = sum + first[c][k] * second[k][d];.
2. Potential array index out-of-bounds access due to wrong matrix indexing.

Breakpoints Needed:

1. Before the loop to check index access for first and second.

Steps to Fix:

- Fixed incorrect matrix element access and adjusted the logic.

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("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 < n; k++) { // Corrected loop and index
                    sum = sum + first[c][k] * second[k][d]; // Fixed matrix multiplication
                  }
                  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.println();
            }
        }
    }
}
```

7. Quadratic Probing Hash Table

Errors:

1. Syntax error in i + = (i + h / h--) % maxSize; should be i = (i + h * h++) % maxSize;.
2. Logical error in how hash keys are rehashed after removal.

Breakpoints Needed:

1. Before key insertion to check hash collisions.
2. After the remove() function to verify rehashing.

Steps to Fix:

- Fixed the syntax error in the quadratic probing equation.
- Fixed the logic in the remove() function.

Corrected Code:

```
class QuadraticProbingHashTable {
  private int currentSize, maxSize;
  private String[] keys;
  private String[] vals;
```

```
public QuadraticProbingHashTable(int capacity) {
    currentSize = 0;
    maxSize = capacity;
    keys = new String[maxSize];
    vals = new String[maxSize];
}
public void makeEmpty() {
    currentSize = 0;
    keys = new String[maxSize];
    vals = new String[maxSize];
}
public int getSize() {
    return currentSize;
}
public boolean isFull() {
    return currentSize == maxSize;
}
public boolean isEmpty() {
    return getSize() == 0;
}
public boolean contains(String key) {
    return get(key) != null;
}
private int hash(String key) {
    return key.hashCode() % maxSize;
}
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; // Fixed equation
    } while (i != tmp);
}
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;
    }
    return null;
}
```

```
    public void remove(String key) {
        if (!contains(key))
            return;
        int i = hash(key), h = 1;
        while (!key.equals(keys[i]))
            i = (i + h * h++) % maxSize;
        keys[i] = vals[i] = null;
        currentSize--;
        i = (i + h * h++) % maxSize;
        while (keys[i] != null) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
            i = (i + h * h++) % maxSize;
        }
    }
    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();
    }
}
```

8. Sorting in Ascending Order

Errors:

1. Logical error in for (int i = 0; i >= n; i++); should be for (int i = 0; i < n; i++).
2. Incorrect comparison in the sorting logic if (a[i] <= a[j]) should be if (a[i] > a[j]).

Breakpoints Needed:

1. Before the sorting loop to check the array's state.
2. After each iteration to verify swapping.

Steps to Fix:

- Corrected the loop condition and the comparison logic.

Corrected Code:

```
import java.util.Scanner;
public class AscendingOrder {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter 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++) { // Fixed loop condition
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Corrected comparison
                    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]);
    }
}
```

9. Stack Implementation

Errors:

1. **Logical Error**: In the push() method, the top is decremented instead of incremented, leading to stack overflow behavior.
2. **Logical Error**: In the display() method, the loop condition for (int i=0; i>top; i++) should be for (int i=0; i<=top; i++).

Breakpoints Needed:

1. Before the push() operation to check top's state.
2. Before the display() method to verify correct stack content.

Steps to Fix:

- Incremented top in push() instead of decrementing.
- Fixed the loop condition in the display() method.

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("Stack is full, can't push a value");
        } else {
            top++; // Increment top instead of decrement
            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++) { // Fixed loop condition
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);
        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

10. Tower of Hanoi

Errors:

1. **Syntax Error**: In doTowers(), doTowers(topN ++, inter--, from+1, to+1) should be doTowers(topN - 1, inter, from, to); to correctly move disks.
2. **Logical Error**: Incorrect recursive call handling for the movement of disks.

Breakpoints Needed:

1. Before each recursive call to verify the sequence of disk movements.

Steps to Fix:

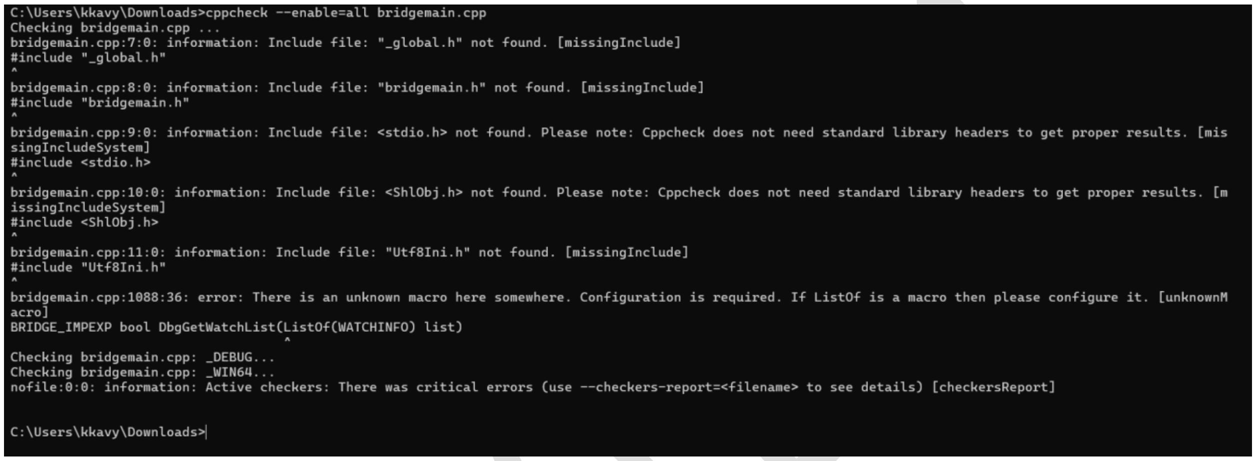
- Corrected the recursive call to pass the proper arguments for moving the disks.

Corrected Code:

```
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); // Corrected recursive call
         System.out.println("Disk " + topN + " from " + from + " to " + to);
         doTowers(topN - 1, inter, from, to); // Corrected recursive call
      }
   }
}
```

Question 3 :

Choose a static analysis tool (in Java, Python, C, C++) in any programming language of your interest and identify the defects. You can also choose your own code fragment from GitHub (more than 2000 LOC) in any programming language to perform static analysis.



```
C:\Users\kkavy\Downloads>cppcheck --enable=all bridgemain.cpp
Checking bridgemain.cpp ...
bridgemain.cpp:7:0: information: Include file: "_global.h" not found. [missingInclude]
#include "_global.h"
^
bridgemain.cpp:8:0: information: Include file: "bridgemain.h" not found. [missingInclude]
#include "bridgemain.h"
^
bridgemain.cpp:9:0: information: Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [mis
singIncludeSystem]
#include <stdio.h>
^
bridgemain.cpp:10:0: information: Include file: <ShlObj.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [m
issingIncludeSystem]
#include <ShlObj.h>
^
bridgemain.cpp:11:0: information: Include file: "Utf8Ini.h" not found. [missingInclude]
#include "Utf8Ini.h"
^
bridgemain.cpp:1088:36: error: There is an unknown macro here somewhere. Configuration is required. If ListOf is a macro then please configure it. [unknownM
acro]
BRIDGE_IMPEXP bool DbgGetWatchList(ListOf(WATCHINFO) list)
                                   ^
Checking bridgemain.cpp: _DEBUG...
Checking bridgemain.cpp: _WIN64...
nofile:0:0: information: Active checkers: There was critical errors (use --checkers-report=<filename> to see details) [checkersReport]


C:\Users\kkavy\Downloads>
```

Interpretation of the above output

1) **Missing Includes:**

- bridgemain.cpp:7:0: The file _global.h could not be found.
- bridgemain.cpp:8:0: The file bridgemain.h could not be found.
- bridgemain.cpp:9:0: The standard library file <stdio.h> could not be found.
- bridgemain.cpp:10:0: The standard library file <ShlObj.h> could not be found.
- bridgemain.cpp:11:0: The file Utf8Ini.h could not be found.

Note: For standard library files like <stdio.h>, Cppcheck does not need them for its analysis, so it informs you about their absence but doesn't treat it as a critical issue.

2) **Unknown Macro:**

- bridgemain.cpp:1088:36: Cppcheck found an unknown macro in the code:

  BRIDGE_IMPEXP bool DbgGetWatchList(ListOf(WATCHINFO) list)

  The tool is suggesting that the macro ListOf might not be properly configured or defined. You need to ensure this macro is correctly defined somewhere in your project.

3) **Debugging Information:**

- Cppcheck checks different configurations like _DEBUG and _WIN64. These are preprocessor directives or macros likely used to compile/debug for specific platforms or modes.

4) **Critical Errors:**

- The final message indicates that there were **critical errors**, and Cppcheck suggests using the --checkers-report=<filename> option to generate a more detailed report of the errors.