Dhirubhai Ambani
Institute of Information and Communication Technology

Software Engineering - IT314

Name: Kashyap Gajera

Student ID : 202201324

Task 1: PROGRAM INSPECTION

❾ Program Inspection for Robin Hood Hashing Code provided in text file:

GitHub Code Link:

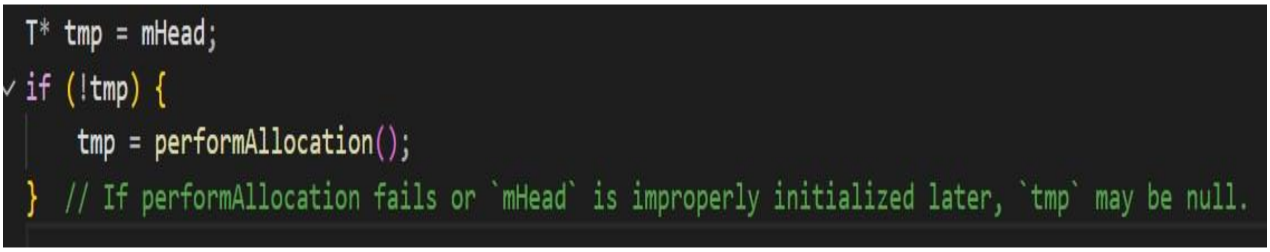
https://github.com/martinus/robin-hood-hashing/blob/master/src/include/robin_hood.h

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

Category A: Data Reference Errors:

1. Uninitialized Variables:

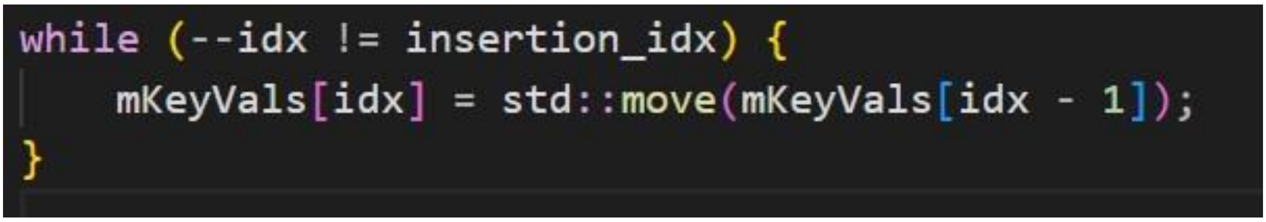
❾ mHead and mListForFree: Initialized to nullptr but not always reset after memory deallocation, leading to potential dangling pointers or uninitialized access.



```
T* tmp = mHead;
if (!tmp) {
    tmp = performAllocation();
} // If performAllocation fails or `mHead` is improperly initialized later, `tmp` may be null.
```

2. Array Bound Violations:

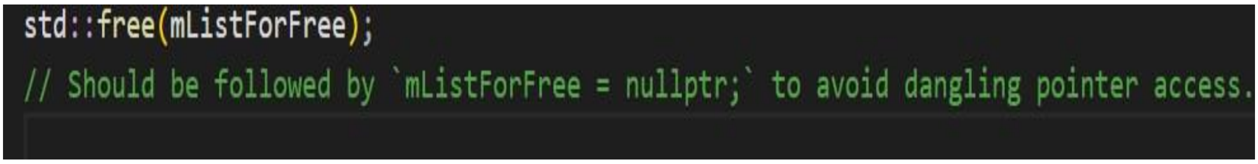
❾ shiftUp and shiftDown operations: No checks ensure that the index is within the array bounds.



```
while (--idx != insertion_idx) {
    mKeyVals[idx] = std::move(mKeyVals[idx - 1]);
}
```

3. Dangling Pointers:

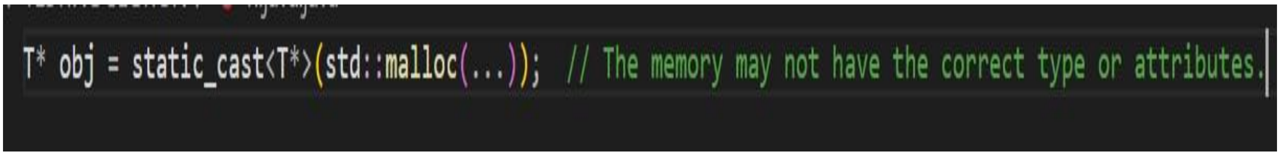
❾ In BulkPoolAllocator: The reset() method frees memory but does not reset the pointer to nullptr.



```
std::free(mListForFree);
// Should be followed by `mListForFree = nullptr;` to avoid dangling pointer access.
```

4. Type Mismatches:

❾ Incorrect Casts in reinterpret_cast_no_cast_align_warning: Casting memory regions without validating types or attributes can lead to subtle bugs.

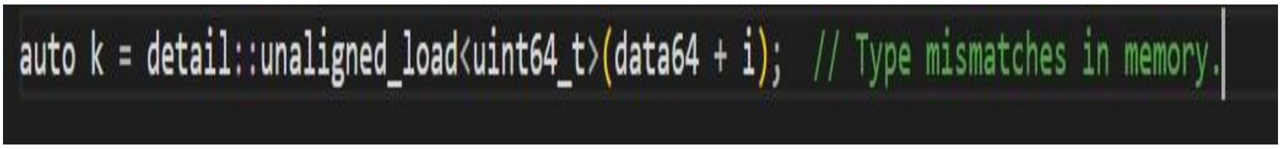


```
T* obj = static_cast<T*>(std::malloc(...));  // The memory may not have the correct type or attributes.
```

Category B: Data-Declaration Errors:

1. Potential Data Type Mismatches:

❾ Casting in hash_bytes: Hashing operations involve multiple castings between data types. If the size or attributes of the data types differ, unexpected behavior can arise.



```
auto k = detail::unaligned_load<uint64_t>(data64 + i);  // Type mismatches in memory.
```

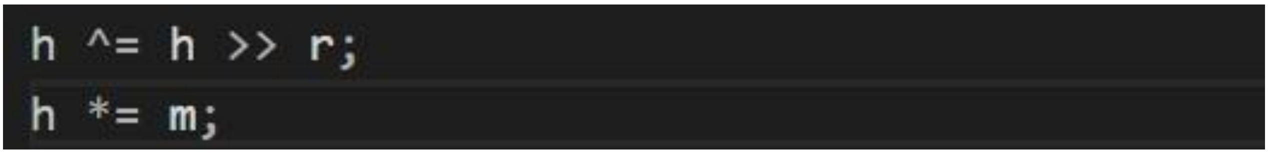
2. Similar Variable Names:

❾ Confusion between similarly named variables: Variables like mHead, mListForFree, and mKeyVals are similar in naming, which could cause confusion during modification or debugging.

Category C: Computation Errors:

1. Integer Overflow:

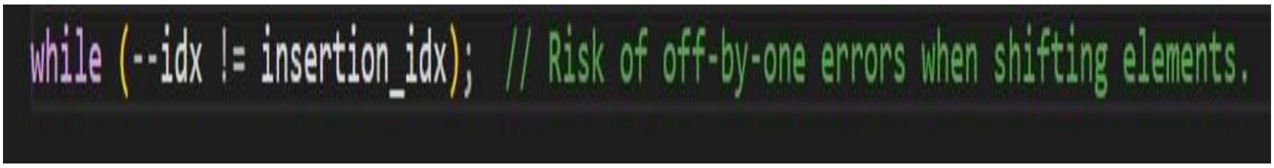
❾ Hash Computations in hash_bytes: The hash function performs multiple shifts and multiplications on large integers, potentially leading to overflow if the result exceeds



```
h ^= h >> r;
h *= m;
```

2. Off-by-One Errors:

❾ Loop Indexing in shiftUp and shiftDown: The loop conditions may result in off-by-one errors, especially if the size of the data structure is mismanaged.

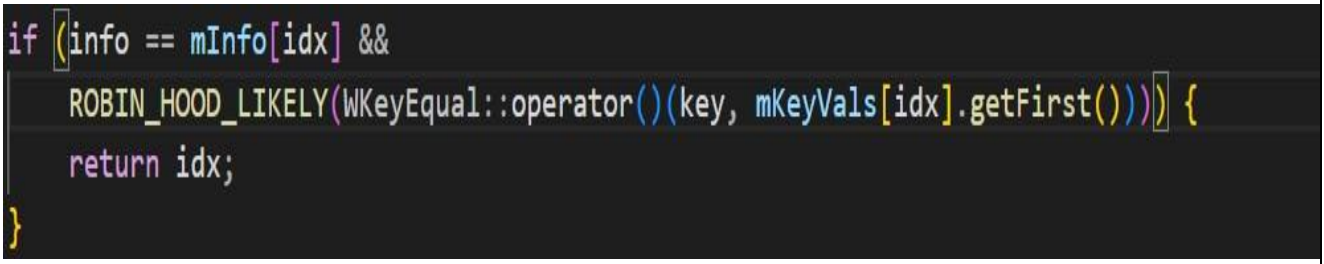


```
while (--idx != insertion_idx); // Risk of off-by-one errors when shifting elements.
```

Category D: Comparison Errors:

1. Incorrect Boolean Comparisons:

❾ In conditions where multiple logical operations are combined, such as in findIdx, improper handling of && and || could lead to incorrect evaluations.



```
if (info == mInfo[idx] &&
    ROBIN_HOOD_LIKELY(WKeyEqual::operator()(key, mKeyVals[idx].getFirst()))) {
    return idx;
}
```

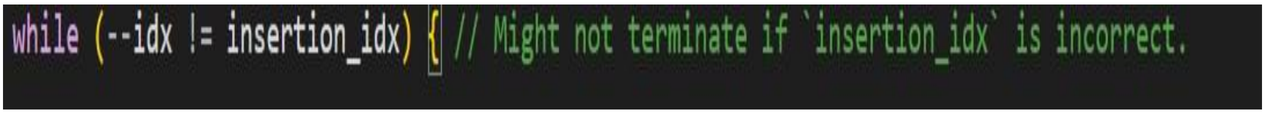
2. Mixed Comparisons:

❾ In some cases, different types (e.g., signed and unsigned integers) are compared, which could lead to incorrect outcomes depending on the system/compiler.

Category E: Control-Flow Errors:

1. Potential Infinite Loop:

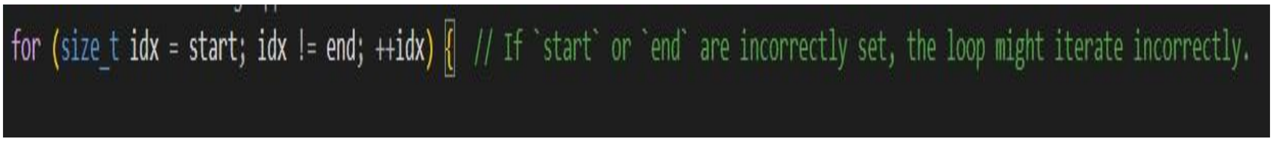
❾ Unterminated Loops: In loops like shiftUp and shiftDown, there is a risk of the loop not terminating correctly if the termination condition is never met.



```
while (--idx != insertion_idx) { // Might not terminate if `insertion_idx` is incorrect.
```

2. Unnecessary Loop Executions:

❾ In some cases, loops might execute one extra time or fail to execute due to incorrect initialization or condition checks.

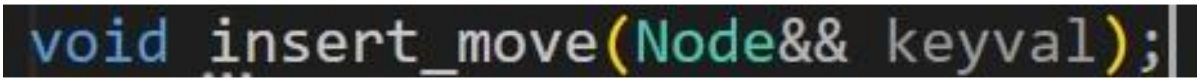


```
for (size_t idx = start; idx != end; ++idx) { // If `start` or `end` are incorrectly set, the loop might iterate incorrectly.
```

Category F: Interface Errors:

1. **Mismatched Parameter Attributes:**

❾ Function Calls: There is potential for parameter mismatch in functions like insert_move. The arguments passed to these functions might not match the expected attributes (e.g., data type, size).



```
void insert_move(Node&& keyval);
```

2. **Global Variables:**

❾ Global variables in different functions: If the same global variable is referenced across different functions or procedures, care must be taken that they are used consistently and initialized properly. This is not explicitly seen but could be a potential error source in expansions of the code.

Category G: Input/Output Errors:

1. **Missing File Handling:**

❾ While the code doesn't deal with files directly, any extension that includes I/O might introduce typical file handling errors such as unclosed files, failure to check for endof-file conditions, or improper error handling.

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

❾ **Category A**: **Data Reference Errors** is the most effective in this case because of the use of manual memory management, pointers, and dynamic data structures. Since errors in pointer dereferencing and memory allocation/deallocation can easily lead to critical issues like crashes, segmentation faults, or memory leaks, focusing on this category is vital. Other important categories are **Computation Errors** and **ControlFlow Errors**, especially for large projects.

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

- **Concurrency Issues**: The inspection does not account for multi-threading or concurrency-related issues, such as race conditions or deadlocks. If this program were expanded to handle multiple threads, issues related to shared resources, locks, and thread safety would need to be addressed.
- **Dynamic Errors**: Some errors, such as those related to memory overflow, underflow, or runtime environment behaviour, may not be caught until the code is executed in a real-world scenario.

4. Is the program inspection technique worth applying?

- **Yes**, the program inspection technique is valuable, particularly for detecting static errors that might not be caught by compilers, such as pointer mismanagement, array bound violations, and improper control flow. Although it may not catch every dynamic issue or concurrency-related bug, it's an essential step to ensure code quality, especially in memory-critical applications like this C++ implementation of hash tables.

This approach improves the code's reliability and helps maintain best practices in memory handling, control flow, and computational logic.

Task 2: CODE DEBUGGIN

- **Code Debugging for given Java files**
- **Note: All the executable files are in separate folder**

1: Armstrong

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

incorrect Calculation of Remainder:

- ❾ The line remainder = num / 10; should be remainder = num % 10; because we want to extract the last digit of the number.

Updating num Incorrectly:

- ❾ The line num = num % 10; should be num = num / 10;. We want to remove the last digit from num after processing it, not take its remainder again.

2. <u>How many breakpoints you need to fix those errors?</u> ❾

Two breakpoints:

1. On the line where the remainder is calculated (remainder = num / 10;).
2. On the line where num is updated (num = num % 10;).

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

- Step 1: Fix the calculation of the remainder to correctly extract the last digit (remainder = num % 10;).
- Step 2: Correctly update num to remove the last digit (num = num / 10;).

<u>Code:</u>

```
class Armstrong {

public static void main(String args[]) {

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

int n = num, check = 0, remainder;

while (num > 0) {

remainder = num % 10;

check += Math.pow(remainder, 3);

num /= 10;

}
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

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

There are two errors in the program:

1. Logical Error in the gcd Method: The condition in the while loop is incorrect. It should be while (a % b != 0) instead of while (a % b == 0). The original condition can lead to an infinite loop if b is not a divisor of a.
2. Logical Error in the lcm Method: The condition to check whether a is a multiple of both x and y is incorrect. It should be if (a % x == 0 && a % y == 0) instead of if (a % x != 0 && a % y != 0).

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

You need two breakpoints to debug and fix the identified errors:

1. A breakpoint at the beginning of the gcd method to monitor the values of a, b, and r.
2. A breakpoint at the beginning of the lcm method to check the initial value of a and how it increments during the loop.

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

1. Fixing the gcd Method:

❾ Changed the condition in the while loop from while (a % b == 0) to while (a % b != 0) to correctly implement the Euclidean algorithm for calculating the GCD.

2. Fixing the lcm Method:

❾ Modified the condition in the if statement from if (a % x != 0 && a % y != 0) to if (a % x == 0 && a % y == 0) to ensure that the method correctly identifies when a is a multiple of both x and y.

Code:

```
import java.util.Scanner;


public class GCD_LCM

{

    static int gcd(int x, int y)

    {

        int r = 0, a, b;

        a = (x > y) ? x : y; // a is greater number

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


        r = b;

        while (a % b != 0) // Corrected condition

        {

            r = a % b;

            a = b;
        b = r;
    }
    return b; // Return the GCD
}

static int lcm(int x, int y)
{
    int a = (x > y) ? x : y; // a is greater number
    while (true)
    {
        if (a % x == 0 && a % y == 0) // Corrected 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

identified.

There are three main errors in the program:

1. Array Indexing Issue: The line int option1 = opt[n++][w]; incorrectly increments n, which can lead to out-of-bounds access in subsequent iterations. It should simply be int option1 = opt[n][w];.
2. Wrong Profit Calculation: In the line int option2 = profit[n-2] + opt[n1][w-weight[n]];, the program incorrectly uses profit[n-2] instead of profit[n] to calculate the profit of the current item.
3. Weight Condition Logic: The condition for taking the item is correct, but the logic for option2 should only be calculated if the item's weight does not exceed the current weight limit (w).

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

You would need three breakpoints to debug and fix the errors:

1. Set a breakpoint at the beginning of the nested loop to check the values of n, w, opt[n][w], and other variables.
2. Set a breakpoint right before the assignment of option1 to monitor how n is changing.
3. Set a breakpoint after the assignment of option2 to verify the calculations for both option1 and option2.

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

1. Correcting Array Indexing:

Changed int option1 = opt[n++][w]; to int option1 = opt[n][w]; to prevent n from being incremented incorrectly.

2. Correcting Profit Calculation:

📫 Modified the line int option2 = profit[n-2] + opt[n-1][w-weight[n]]; to int option2 = profit[n] + opt[n-1][w-weight[n]]; to reference the correct item profit.

3. Adjusting Weight Condition Logic:

❾ Added a condition to ensure that option2 is only calculated if the current item's weight does not exceed w. This prevents erroneous profit calculations for items that can't be added.

CODE:

```
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 - 1][w]; // Corrected to use n-1

      // Take item n
      int option2 = Integer.MIN_VALUE;
      if (weight[n] <= w) { // Check if we can take item n
        option2 = profit[n] + opt[n - 1][w - weight[n]]; // Corrected profit reference
      }

      // 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 -= weight[n];
            } else {
                take[n] = false;
            }
        }

        // Print results
        System.out.println("Item\tProfit\tWeight\tTake");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
        }
    }
}
```

4: Magic Number

identified.

There are four errors in the program:

1. Logical Error in the Inner Loop: The condition in the line while(sum==0) should be while(sum!=0). The current condition will not enter the loop when sum is zero, which is incorrect.
2. Incorrect Calculation in the Inner Loop: The line s=s*(sum/10); should be s = s + (sum % 10); to correctly accumulate the sum of the digits.
3. Missing Semicolon: The line sum=sum%10 should have a semicolon at the end: sum = sum % 10;.
4. Logical Error in the While Loop: The outer loop condition while(num>9) should be while(num>9 || num == 0) to account for the scenario where the number becomes zero.

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

You would need three breakpoints to effectively debug and fix the errors:

1. Set a breakpoint at the beginning of the inner loop to observe the values of sum and s.
2. Set a breakpoint at the beginning of the outer loop to check the current value of num.
3. Set a breakpoint before the final if statement to verify the final value of num before making the magic number determination.

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

1. Correcting the Inner Loop Condition:

❾ Changed while(sum==0) to while(sum!=0) to ensure the loop iterates while there are digits left to process.

2. Fixing the Digit Summation Logic:

❾ Updated the line s=s*(sum/10); to s = s + (sum % 10); to accumulate the digits correctly.

3. Adding Missing Semicolon:

❾ Added a semicolon at the end of sum = sum % 10;.

4. Adjusting the Outer Loop Condition:

❾ Changed the outer loop condition from while(num>9) to while(num > 9 || num == 0) to handle the case where num might reduce to zero.

CODE:

```
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 || num == 0) { // Corrected condition
            sum = num;
            int s = 0;
            while (sum != 0) { // Corrected condition
                s = s + (sum % 10); // Corrected digit summation logic
                sum = sum / 10; // Corrected to divide by 10
            }
            num = s; // Update num to the sum of digits
        }
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
        ob.close(); // Close the scanner
    }
}
```

5: Merge Sort

identified.

There are four main errors in the program:

1. Incorrect Array Slicing: The lines int[] left = leftHalf(array + 1); and int[] right = rightHalf(array - 1); are incorrect because you cannot slice arrays by adding or subtracting integers. It should be splitting the array into halves correctly.
2. Incorrect Parameters in Recursive Calls: When calling merge(array, left++, right--);, you cannot use the increment/decrement operators (++ and --) on the arrays. You should pass the arrays as is.
3. Incorrect Calculation of Left and Right Sizes: The size calculation in leftHalf and rightHalf should account for the entire array. The size for the left half is (array.length + 1) / 2 to correctly handle odd lengths.
4. Missing Merging Logic: In the merge method, the original array (result) should not be passed in the manner shown. Instead, it should be the original array passed to the merge sort function which gets modified. This logic needs to be integrated properly.

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

You would need three breakpoints to effectively debug and fix the errors:

1. Set a breakpoint at the beginning of the mergeSort method to inspect how the array is being split and what the left and right halves are.
2. Set a breakpoint before the merge operation to check the contents of the left and right arrays.
3. Set a breakpoint inside the merge method to see how elements are being merged back into the original array.

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

1. Correcting Array Slicing:
   - Instead of int[] left = leftHalf(array + 1); and int[] right = rightHalf(array - 1);, change it to correctly split the array using Arrays.copyOfRange.
2. Fixing Parameters in Recursive Calls:
   - Update the call to merge by passing the arrays without using the increment/decrement operators: merge(array, left, right);.
3. Adjusting Size Calculations:
   - Change the size calculation in leftHalf and rightHalf methods to (array.length + 1) / 2 for the left half and the rest for the right half.
4. Merging Logic:
   - Ensure that the merge method correctly combines the sorted arrays back into the original array.

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));
    }

    // 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 mid = (array.length + 1) / 2; // Calculate the midpoint
            int[] left = Arrays.copyOfRange(array, 0, mid);
            int[] right = Arrays.copyOfRange(array, mid, array.length);

            // recursively sort the two halves
            mergeSort(left);
            mergeSort(right);

            // merge the sorted halves into a sorted whole
            merge(array, left, right);
        }
    }
    // Merges the given left and right arrays into the given
    // result array.
    // pre : result is empty; left/right are sorted
    // post: result contains result of merging sorted lists;
    public static void merge(int[] result,
                             int[] left, int[] right) {
        int i1 = 0;   // index into left array
        int i2 = 0;   // index into right array

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length &&
                    left[i1] <= right[i2])) {
                result[i] = left[i1];    // take from left
                i1++;
            } else {
                result[i] = right[i2];   // take from right
                i2++;
            }
        }
    }
}
```

6: Multiply metrics

identified.

There are five main errors in the program:

1. Array Indexing Errors: In the line sum = sum + first[c-1][c-k] * second[k1][k-d];, the indices c-1 and k-d are incorrect. They should use c and k for proper indexing since the matrix elements start from index 0.
2. Uninitialized Variables: The variable sum is being reused without resetting in the inner loop properly. This can lead to incorrect calculations in subsequent iterations. It should be reset to 0 at the start of each c and d iteration.
3. Wrong Output Input Prompt: The input prompt for the second matrix incorrectly states, “Enter the number of rows and columns of first matrix” instead of “Enter the number of rows and columns of second matrix”.
4. Multiplication Logic Issue: The multiplication logic needs to access elements of the matrices correctly. The correct formula for matrix multiplication is first[c][k] * second[k][d].
5. Potential Readability Issue: The output formatting is slightly misleading, as it shows the product matrix but doesn't include a proper header or format.

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

You would need three breakpoints to effectively debug and fix the errors:

1. Set a breakpoint inside the multiplication loop to inspect the indices and the values being multiplied.
2. Set a breakpoint before the printing of the multiplication results to check the contents of the multiply array.
3. Set a breakpoint after reading the second matrix to verify that the inputs are being read correctly.

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

1. Correcting Array Indexing:

❾ Change sum = sum + first[c-1][c-k] * second[k-1][k-d]; to sum = sum + first[c][k] * second[k][d]; to correctly access the elements of the matrices.

2. Resetting Variables:

❾ Move the reset of the sum variable to the beginning of the inner loop for d to ensure it starts fresh for each element calculation: sum = 0; should be at the start of the for (d = 0; d < q; d++) loop.

3. Fixing Input Prompts:

❾ Update the prompt for the second matrix to say “Enter the number of rows and columns of the second matrix”.

4. Adjusting Output Formatting:

❾ Consider adding headers to clarify that the following output is the product matrix.

```
import java.util.Scanner;


class MatrixMultiplication {
    public static void main(String args[]) {
        int m, n, p, q;


        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of first matrix");
        m = in.nextInt();
        n = in.nextInt();


        int first[][] = new int[m][n];
```

```
  System.out.println("Enter the elements of first matrix");

  for (int c = 0; c < m; c++)
      for (int d = 0; d < n; d++)
          first[c][d] = in.nextInt();

  System.out.println("Enter the number of rows and columns of second matrix");
  p = in.nextInt();
  q = in.nextInt();

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

      for (int c = 0; c < m; c++) {
          for (int d = 0; d < q; d++) {
              int sum = 0; // Reset sum at the start of each multiplication
          for (int k = 0; k < n; k++) {
            sum = sum + first[c][k] * second[k][d]; // Corrected
   multiplication logic
          }

          multiply[c][d] = sum; // Store the result
        }
      }

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

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

        System.out.print("\n");
      }
    }
    in.close(); // Close the scanner
  }
}
```

7: Quadratic Probing

identified.

There are several errors in the program:

1. Syntax Error in the Insert Method: The line i + = (i + h / h--) % maxSize; contains a space in the += operator, causing a compilation error.
2. Incorrect Hashing Logic: The line i = (i + h * h++) % maxSize; is incorrect because it modifies h within the loop, which can lead to an infinite loop.
3. Key Removal Logic: In the remove method, currentSize-- is decremented twice, which results in incorrect size management.
4. Uninitialized Value Printing: When printing the hash table, the output might include null values or improperly formatted outputs.
5. Clear Method Logic: The makeEmpty method does not clear the actual objects in the arrays, leading to potential memory issues.

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

To fix these errors, you would need the following breakpoints:

1. Breakpoint on the Insert Method: Before the line containing the i + = operator to check the current value of i.
2. Breakpoint on the Hash Method: To observe how the hash value is calculated for different keys.
3. Breakpoint on the Remove Method: To ensure the correct key is being removed and to check the state of the hash table after the removal.
4. Breakpoint in the Print Method: To validate the correct values are being printed from the hash table.

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

1. Correcting the Insert Method: Remove the space in the += operator and correct the logic for incrementing h.
2. Fixing the Hash Method: Ensure that the hashing algorithm doesn't modify h directly and doesn't lead to an infinite loop.

3. Updating Removal Logic: Adjust the remove method to ensure currentSize is only decremented once after a successful removal.
4. Enhancing Print Logic: Add checks to avoid printing null values and ensure that the output format is clear.
5. Adjusting the Make Empty Logic: Modify the makeEmpty method to reset the actual contents of the keys and values arrays.

CODE:

```
import java.util.Scanner;

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 Math.abs(key.hashCode()) % maxSize; // Ensuring a positive index
    }

    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 increment
        h++;
    } 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;
        h++;
    }
```

```
        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;

        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--;
    }

    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();
    }
}

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");
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());

        char ch;
        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;
            }
            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');
    }
}
```

8: Sorting Array

identified.

There are several errors in the program:

1. Class Name Error: The class name Ascending _Order contains a space, which is not allowed in Java. It should be AscendingOrder.
2. Incorrect Loop Condition: The outer loop for (int i = 0; i >= n; i++); has an incorrect condition (i >= n), which will cause it to never execute. The correct condition should be i < n.
3. Unnecessary Semicolon: There is an unnecessary semicolon at the end of the outer loop declaration (for (int i = 0; i >= n; i++);), which ends the loop prematurely.
4. Sorting Logic: The comparison in the sorting condition is incorrect. It should be if (a[i] > a[j]) to ensure that the smaller number is placed before the larger number.
5. Output Formatting: The final output will have an extra comma if the elements are printed directly. It should be formatted correctly to avoid trailing commas.

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

To fix these errors, you would need the following breakpoints:

1. Breakpoint on Class Declaration: To check the correct naming of the class.
2. Breakpoint on Outer Loop: To observe the initial value of i and ensure that the loop condition is correct.
3. Breakpoint on Sorting Logic: To validate the values of a[i] and a[j] before and after swapping.
4. Breakpoint on Output: To check the formatting of the output and ensure it doesn't include unwanted commas.

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

1. Renaming the Class: Change the class name from Ascending _Order to AscendingOrder.
2. Correcting the Loop Condition: Change the loop condition from i >= n to i < n.
3. Removing the Semicolon: Remove the unnecessary semicolon after the outer loop declaration.
4. Fixing the Sorting Logic: Change the condition in the sorting logic to if (a[i] > a[j]).
5. Formatting the Output: Update the output logic to avoid trailing commas.
6.

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++) { // Corrected condition
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Fixed sorting logic
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }

        System.out.print("Ascending Order: ");
        for (int i = 0; i < n; i++) {
            System.out.print(a[i]);
            if (i < n - 1) { // Avoid trailing comma
                System.out.print(", ");
            }
        }
        System.out.println(); // Add newline for cleaner output
    }
}
```

9: Stack Implementation

identified.

There are several errors in the program:

1. Incorrect Logic in push Method: The line top--; should be top++; because we want to increment the top index to push the value onto the stack.
2. Incorrect Logic in pop Method: The line top++; should be top--; because we want to decrement the top index to remove the top element of the stack.
3. Incorrect Condition in display Method: The loop condition for (int i = 0; i > top; i++) is incorrect. It should be i <= top to ensure all elements in the stack are displayed.
4. Handling Stack Underflow: The pop method should return the popped value. This can be done by storing the value being popped before decrementing top.
5. Displaying the Stack Contents: The output format may be misleading because the elements are not displayed correctly after popping.

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

To fix these errors, you would need the following breakpoints:

1. Breakpoint on push Method: To check the value of top before and after the increment.
2. Breakpoint on pop Method: To observe the value being popped and the state of top.
3. Breakpoint on display Method: To verify the loop condition and ensure all elements are printed correctly.

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

1. Corrected Logic in push Method: Change top--; to top++; so that the next element is added at the correct index.
2. Corrected Logic in pop Method: Change top++; to top--; to ensure the top element is correctly removed from the stack.

3. Updated Loop Condition in display Method: Change i > top to i <= top so that all elements in the stack are displayed.
4. Return Value in pop Method: Modify the pop method to return the value that was popped from the stack.
5. Adjust the Display Logic: Ensure the display method properly reflects the current state of the stack after popping elements.

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++; // Corrected increment for push
            stack[top] = value;
        }
```

```
}

public int pop() {
    if (!isEmpty()) {
        int poppedValue = stack[top]; // Get the value to return
        top--; // Corrected decrement for pop
        return poppedValue; // Return the popped value
    } else {
        System.out.println("Can't pop...stack is empty");
        return -1; // Indicate that the stack is empty
    }
}

public boolean isEmpty() {
    return top == -1;
}

public void display() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
        return;
    }
    System.out.print("Stack elements: ");
    for (int i = 0; i <= top; i++) { // Corrected 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

identified.

There are several errors in the program:

1. Incorrect Increment and Decrement in Recursive Call: The line doTowers(topN ++, inter--, from+1, to+1) is incorrect. The post-increment and post-decrement operators (++ and --) are used incorrectly in this context. They should not be used this way, as they do not modify the values passed to the function.
2. Missing Recursive Call for Disk Movement: The logic for handling disk movements in the recursive calls is not accurate, leading to incorrect calculations.
3. Printing Issues: The final output does not match the expected movements of the disks correctly due to the incorrect handling of parameters.

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

You would need the following breakpoints to fix the errors:

1. Breakpoint on the first doTowers call: To check the values of topN, from, inter, and to before executing the recursive calls.
2. Breakpoint before the printing statement: To observe the correct flow of disk movements.
3. Breakpoint on the second doTowers call: To ensure the parameters are being correctly passed after the first recursive call.

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

1. Corrected Recursive Call: Change doTowers(topN ++, inter--, from+1, to+1) to doTowers(topN - 1, inter, from, to) in the recursive call for moving the remaining disks.
2. Removed Invalid Modifications: Ensure that the values for from, inter, and to are not modified with post-increment and post-decrement operators. Instead, pass the original variables directly.

3. Clarified Disk Movement Logic: Ensure that the recursive logic correctly follows the Tower of Hanoi algorithm.

CODE:

```
public class MainClass {
  public static void main(String[] args) {
    int nDisks = 3; // Number of disks
    doTowers(nDisks, 'A', 'B', 'C'); // A = source, B = auxiliary, C = destination
  }


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

Task 3: Static Tool Analysis

Using cppcheck, I run static analysis tool for 1300 lines of code used above for program inspection.

Results:

```
[202201324_Lab3_2.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:10]: (information) Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_2.c:0]: (information) Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches.

[202201324_Lab3_2.c:116]: (warning) scanf() without field width limits can crash with huge input data.

[202201324_Lab3_2.c:120]: (warning) scanf() without field width limits can crash with huge input data.

[202201324_Lab3_2.c:126]: (warning) scanf() without field width limits can crash with huge input data.

[202201324_Lab3_2.c:127]: (warning) scanf() without field width limits can crash with huge input data.
```

```
[202201324_Lab3_2.c:133]: (warning) scanf() without field width limits can crash with huge input data.

[202201324_Lab3_2.c:34]: (style) The scope of the variable 'ch' can be reduced.

[202201324_Lab3_2.c:115]: (style) The scope of the variable 'path2' can be reduced.

[202201324_Lab3_2.c:16]: (style) Parameter 'file' can be declared as pointer to const

[202201324_Lab3_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const

[202201324_Lab3_2.c:40]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.

[202201324_Lab3_3.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_3.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_3.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_3.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_Lab3_3.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201324_lab3_1.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201324_lab3_1.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
```

```
[202201324_lab3_1.c:29]: (style) The scope of the variable 'ch' can be reduced.

[202201324_lab3_1.c:11]: (style) Parameter 'file' can be declared as pointer to const

[202201324_lab3_1.c:50]: (style) Variable 'direntp' can be declared as pointer to
const

[202201324_lab3_1.c:35]: (warning) Storing fgetc() return value in char variable and
then comparing with EOF.

[Covid-Management-System.cpp:4]: (information) Include file: <iostream> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:5]: (information) Include file: <cstring> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:6]: (information) Include file: <windows.h> not
found. Please note: Cppcheck does not need standard library headers to get proper
results. [Covid-Management-System.cpp:7]: (information) Include file: <fstream> not
found.
Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:8]: (information) Include file: <conio.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:9]: (information) Include file: <iomanip> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:10]: (information) Include file: <cstdlib> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:11]: (information) Include file: <string> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:12]: (information) Include file: <unistd.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:562]: (portability) fflush() called on input stream
'stdin' may result in undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:565]: (portability) fflush() called on input stream
'stdin' may result in undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:614]: (portability) fflush() called on input stream
'stdin' may result in undefined behaviour on non-linux systems. [Covid-Management-
System.cpp:1121]: (portability) fflush() called on input stream 'stdin' may result in
undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:538]: (style) C-style pointer casting

[Covid-Management-System.cpp:619]: (style) C-style pointer casting
```

[Covid-Management-System.cpp:641]: (style) C-style pointer casting

[Covid-Management-System.cpp:646]: (style) C-style pointer casting

[Covid-Management-System.cpp:749]: (style) C-style pointer casting

[Covid-Management-System.cpp:758]: (style) C-style pointer casting

[Covid-Management-System.cpp:788]: (style) C-style pointer casting

[Covid-Management-System.cpp:797]: (style) C-style pointer casting

[Covid-Management-System.cpp:827]: (style) C-style pointer casting

[Covid-Management-System.cpp:836]: (style) C-style pointer casting

[Covid-Management-System.cpp:866]: (style) C-style pointer casting

[Covid-Management-System.cpp:875]: (style) C-style pointer casting

[Covid-Management-System.cpp:907]: (style) C-style pointer casting

[Covid-Management-System.cpp:973]: (style) C-style pointer casting

[Covid-Management-System.cpp:982]: (style) C-style pointer casting

[Covid-Management-System.cpp:1012]: (style) C-style pointer casting

[Covid-Management-System.cpp:1021]: (style) C-style pointer casting

[Covid-Management-System.cpp:1051]: (style) C-style pointer casting

[Covid-Management-System.cpp:1060]: (style) C-style pointer casting

[Covid-Management-System.cpp:1090]: (style) C-style pointer casting

[Covid-Management-System.cpp:1099]: (style) C-style pointer casting

[Covid-Management-System.cpp:1181]: (style) C-style pointer casting

[Covid-Management-System.cpp:1207]: (style) C-style pointer casting

[Covid-Management-System.cpp:1216]: (style) C-style pointer casting

[Covid-Management-System.cpp:1307]: (style) C-style pointer casting

[Covid-Management-System.cpp:1317]: (style) C-style pointer casting [Covid-Management-System.cpp:1320]: (style) C-style pointer casting

[Covid-Management-System.cpp:427]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

[Covid-Management-System.cpp:443]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

[Covid-Management-System.cpp:459]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

```
[Covid-Management-System.cpp:892]: (style) Consecutive return, break, continue,
goto or throw statements are unnecessary.

[Covid-Management-System.cpp:306]: (style) The scope of the variable 'usern' can
be reduced.

[Covid-Management-System.cpp:48] -> [Covid-Management-System.cpp:277]:
(style)
Local variable 'user' shadows outer function

[Covid-Management-System.cpp:40] -> [Covid-Management-System.cpp:304]:
(style) Local variable 'c' shadows outer variable

[Covid-Management-System.cpp:275]: (performance) Function parameter 'str' should
be passed by const reference.

[Covid-Management-System.cpp:277]: (style) Unused variable: user

[Covid-Management-System.cpp:304]: (style) Unused variable: c
```