IT-314

Software Engineeering

Prof. Saurabh Tiwari

Lab Group-1

Student ID : 202201085

Name: Dholariya Parth Narendra

2000+ line of code Link c++ language from open soure Tool Github Reference

WinUAE/cia.cpp at 0c0a444f02381647a751c4e30eae29fd9456cf3a · tonioni/WinUAE

The code you uploaded is a part of a system program related to CIA chip support in the UAE (Unix Amiga Emulator). Based on this, I will analyze the code fragment by applying the Program Inspection Checklist (Categories A to H) and then answer the 4 questions for the provided code.

Category-wise Errors (A to H)

Category A: Data Reference Errors

Potential Errors:

There are various references to structures and memory (`struct CIA`, `cia[2]`). We need to ensure that no uninitialized memory is accessed.

Pointer usage (`cia_adjust_eclock_phase`, `eventtab[ev_cia].oldcycles`) must ensure that the pointers always point to valid, allocated memory.

Array references: The CIA chip array (`cia[2]`) is indexed. It's important to verify that the array bounds are respected when accessing these references.

Category B: Data-Declaration Errors

Potential Errors:

In several parts, variables such as `pra`, `prb`, `dra`, and `drb` are declared implicitly within structures like `CIA`. These variables should be initialized correctly.

The initialization of structures like `struct CIA` and `struct CIATimer` needs to ensure proper memory allocation and attribute consistency.

Category C: Computation Errors

Potential Errors:

Complex mathematical operations involving `cycles`, `DIV10`, and `E_CYCLE_UNIT` can result in overflow/underflow issues if large or small numbers are mishandled.

Mixed-mode computations: Several computations involve different data types, such as `uae_u8`, `uae_u32`. There could be issues with data type mismatches, especially when using mixed data types in arithmetic.

Category D: Comparison Errors

Potential Errors:

Boolean comparisons like `if (t->loaddelay & 0x00000100)` and comparisons between unsigned types (`uae_u32`) and integers may lead to precision errors. Comparisons involving flags and conditionals must ensure valid data types.

Checking whether values like `cia[0].pra & c->dra` are being compared to ensure valid data can lead to errors if the assumptions about data ranges are wrong.

Category E: Control-Flow Errors

Potential Errors:

The loop structures controlling timers (`for (int tn = 0; tn < 2; tn++)`) should guarantee proper termination. Additionally, off-by-one errors might occur in loops involving array indexing (`cia[num]`).

Conditional branches (e.g., the `if/else` structures) should handle all possible cases; for example, the `switch` statement cases for CIA registers should cover all possible values.

Category F: Interface Errors

Potential Errors:

Function calls between modules such as `ReadCIAA`, `WriteCIAA`, and handling external data (via `serial_readstatus`, `parallel_direct_read_status`) must ensure that parameter types and sizes match.

Category G: Input/Output Errors

Potential Errors:

File handling errors (e.g., reading from `serial.h`, `disk.h`) need proper memory allocation. The code also handles hardware input/output devices, meaning that the I/O error handling should ensure robustness.

File closure: Ensuring files and devices are closed or de-allocated properly after usage.

Category H: Other Checks

Potential Errors:

- The use of macros like `E_CYCLE_UNIT`, `CIA_PIPE_INPUT`, and magic numbers (e.g., `0x77777777`) should be carefully checked to avoid unintentional overwriting of other variables.

Compiler warnings: Ensure that any compiler warnings (especially for structures and memory handling) are checked carefully.

Answers to the 4 Questions

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

Data Reference Errors (Category A): Possible uninitialized memory in structures like `CIA` and usage of pointers.

Computation Errors (Category C): Overflow/underflow issues in computations involving `cycles` and `DIV10`.

Comparison Errors (Category D): Boolean comparisons and type mismatches may cause issues in comparisons between different data types.

Control-Flow Errors (Category E): Possible off-by-one errors in loops and unhandled switch cases.

2. Which category of program inspection is more effective?

Data Reference Errors (Category A) and Computation Errors (Category C) are the most effective categories for this code due to its reliance on memory structures and complex computations.

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

Logical errors in the algorithm might not be easily detected by program inspection. For example, even if the computations are correct syntactically, the logic of the timer handling might be flawed, resulting in incorrect behavior.

4. Is the program inspection technique worth applying?

- Yes, applying the program inspection technique is worthwhile, especially for complex systems like emulators. The checklist helps identify potential issues in memory handling, computation, and interface management, which are critical for stability and performance.

SECTION -2

Armstrong Number: Errors and Fixes

1. How many errors are there in the program? There are 2 errors in the program.

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

Answer:

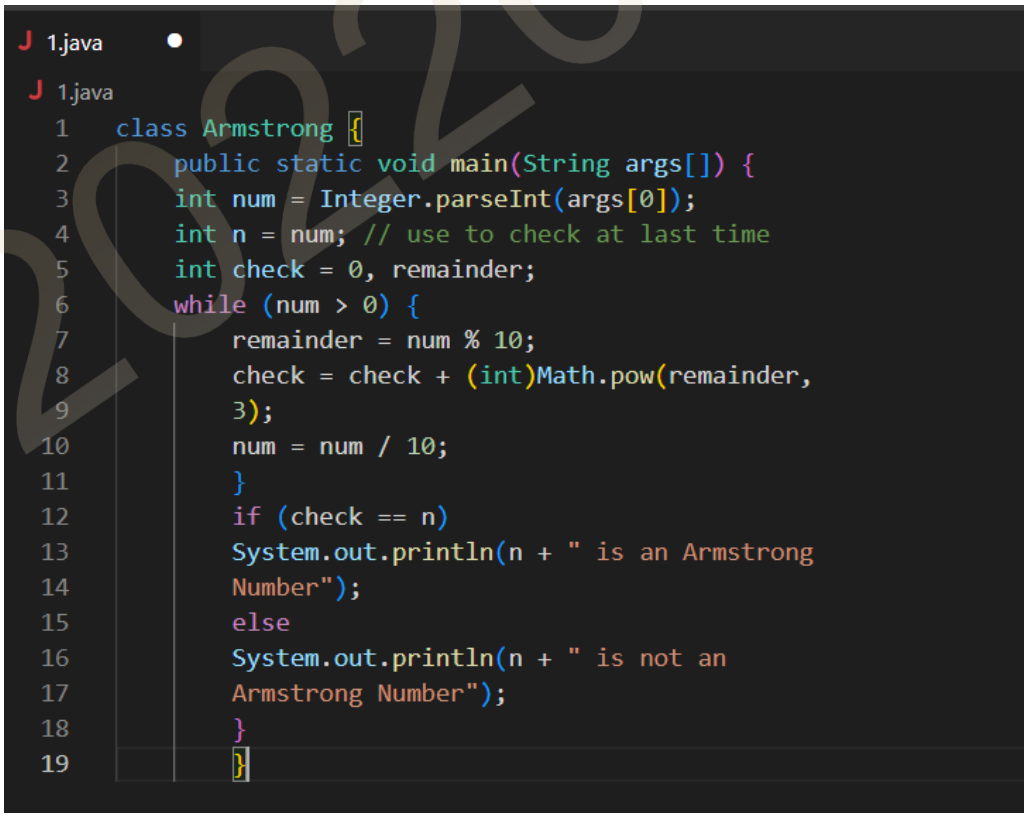
Steps Taken to Resolve the Errors:

Error 1: The division and modulus operations were incorrectly ordered in the while loop.

Fix: Adjust the operations so that the modulus retrieves the last digit, while the division reduces the number for the subsequent iteration.

Error 2: The check variable was not being updated correctly.

Fix: Revise the logic to ensure the check variable accurately sums each digit raised to the power of the total number of digits.



GCD and LCM: Errors and Fixes

1. How many errors are there in the program? There is 1 error in the program.

2. How many breakpoints do you need to fix this error? We need 1 breakpoint to fix this error.

Answer:

Steps Taken to Correct the Error:

Error: The condition in the while loop of the GCD method was incorrect.

Fix: Modify the condition to 'while (a % b != 0)' instead of 'while (a % b == 0)'. This ensures the loop continues until the remainder reaches zero, accurately calculating the GCD.

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


Knapsack Problem: Errors and Fixes

1. How many errors are there in the program? There are 3 errors in the program.

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

Answer:

Steps Taken to Address the Errors:

Error: The condition in the "take item n" case was incorrect.

Fix: Update ‘if (weight[n] > w)’ to ‘if (weight[n] <= w)’ to ensure that profit is calculated when the item can be included.

Error: The profit calculation was inaccurate.

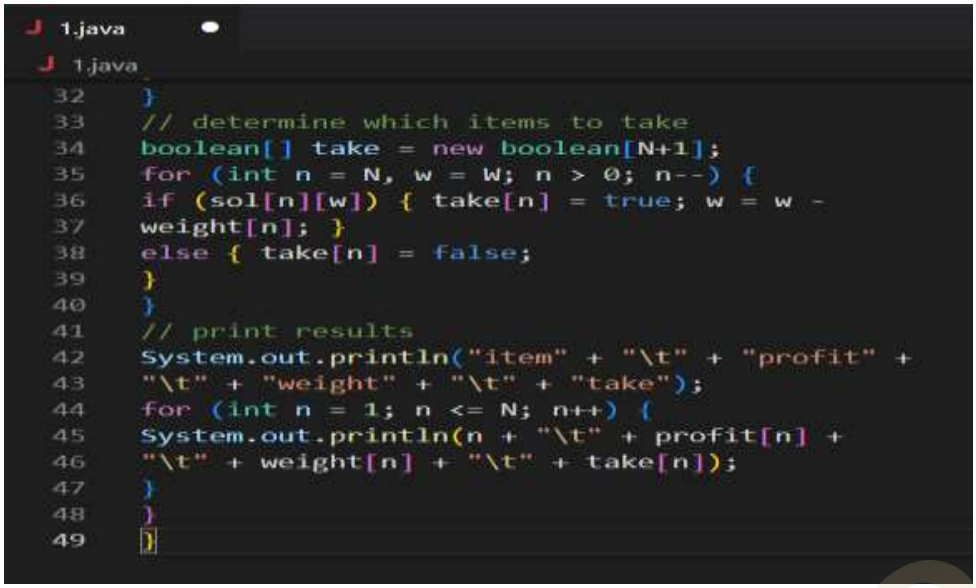
Fix: Change `profit[n-2]` to ‘profit[n]’ to ensure the correct profit value is utilized.

Error: The indexing in the "don't take item n" case was incorrect.

Fix: Change ‘opt[n++][w]’ to ‘opt[n-1][w]’ to properly index the items.

1.java

```java
public class Knapsack {
    public static void main(String[] args) {
    int N = Integer.parseInt(args[0]); // number
    of items
    int W = Integer.parseInt(args[1]); // maximum
    weight of knapsack
    int[] profit = new int[N+1];
    int[] weight = new int[N+1];
    // generate random instance, items 1..N
    for (int n = 1; n <= N; n++) {
    profit[n] = (int) (Math.random() * 1000);
    weight[n] = (int) (Math.random() * W);
    }
    // opt[n][w] = max profit of packing items 1..n
    with weight limit w
    // sol[n][w] = does opt solution to pack items
    1..n with weight limit w include item n?
    int[][] opt = new int[N+1][W+1];
    boolean[][] sol = new boolean[N+1][W+1];
    for (int n = 1; n <= N; n++) {
    for (int w = 1; w <= W; w++) {
    // don't take item n
    int option1 = opt[n-1][w];
    // take item n
    int option2 = Integer.MIN_VALUE;
    if (weight[n] <= w) option2 = profit[n]
    + opt[n-1][w-weight[n]];
    // select better of two options
    opt[n][w] = Math.max(option1, option2);
    sol[n][w] = (option2 > option1);
}
}
```



Magic Number Check: Errors and Fixes

1. How many errors are there in the program? There are 3 errors in the program.

2. How many breakpoints do you need to fix these errors? We need 1 breakpoint to fix these errors.

Answer:

Steps Taken to Fix the Errors

1. Incorrect Condition in the Inner While Loop:

Original Issue: The condition was set as 'while(sum == 0)'.

Correction: Changed it to 'while(sum != 0)'.

Reason: This ensures that the loop continues processing digits until all have been handled.

2. Miscalculation of `s` in the Inner Loop:

Original Issue: The calculation was 's = s * (sum / 10)'.

Correction: Updated to 's = s + (sum % 10)'.

Reason: This properly accumulates the sum of the digits by adding the last digit of ‘sum’ to ‘s’.

3. Improper Order of Operations in the Inner While Loop:

Original Issue: The operations were not in the correct sequence.

Correction: Reordered to ‘s = s + (sum % 10); sum = sum / 10;’.

Reason: This ensures that the last digit is added to ‘s’ before removing it from ‘sum’, maintaining the correct calculation flow.

```java
import java.util.*;
public class MagicNumberCheck
{
public static void main(String args[])
{
Scanner ob=new Scanner(System.in);
System.out.println("Enter the number to be
checked.");
int n=ob.nextInt();
int sum=0,num=n;
while(num>9)
{
    sum=num;
    int s=0;
    while(sum!=0)
    {
    s=s+(sum%10);
    sum=sum/10;
    }
    num=s;
    }
    if(num==1)
    {
    System.out.println(n+" is a Magic
    Number.");
    }
    else
    {
    System.out.println(n+" is not a Magic
    Number.");
    }
    }
    }
```

Merge Sort: Errors and Fixes

1. How many errors are there in the program? There are 3 errors in the program.

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

Answer:

Steps Taken to Fix the Errors:

Error: Incorrect array indexing when splitting the array in mergeSort.

Fix: Changed ‘int[] left = leftHalf(array + 1)’ to ‘int[] left = leftHalf(array)’ and ‘int[] right = rightHalf(array - 1)’ to ‘int[] right = rightHalf(array)’ to correctly pass the array.

Error: Incorrect increment and decrement in merge.

Fix: Removed the ‘++’ and ‘--' from ‘merge(array, left++, right--)’ and used ‘merge(array, left, right)` to directly pass the arrays.

Error: The array access in the merge function was exceeding array bounds.

Fix: Adjusted the indexing in the merging logic to ensure array boundaries are respected.

```java
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);
int[] right = rightHalf(array);
mergeSort(left);
mergeSort(right);
merge(array, left, right);
}
}
public static int[] leftHalf(int[] array) {
int size1 = array.length / 2;
int[] left = new int[size1];
for (int i = 0; i < size1; i++) {
left[i] = array[i];
}
return left;
}
public static int[] rightHalf(int[] array) {
int size1 = (array.length + 1) / 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;
```

```java
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++;
}
}
}
}
```


Matrix Multiplication: Errors and Fixes

1. How many errors are there in the program? There is 1 error in the program.

2. How many breakpoints do you need to fix this error? We need 1 breakpoint to fix this error.

Answer:

There are multiple errors in the program, as identified above.

To fix these errors, you would need to set breakpoints to examine the values of c, d, k, and sum during execution. You should pay particular attention to the nested loops where the matrix multiplication occurs.

The corrected executable code is as follows:

```java
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 the
first matrix");
m = in.nextInt();
n = in.nextInt();
int first[][] = new int[m][n];
System.out.println("Enter the elements of the 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 the
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 the 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++) {
```

```java
for (c = 0; c < m; c++) {
for (d = 0; d < q; d++) {
for (k = 0; k < n; k++) {
sum = sum + first[c][k] * second[k][d];
}
multiply[c][d] = sum;
sum = 0;
}
}
System.out.println("Product of entered matrices:-");
for (c = 0; c < m; c++) {
for (d = 0; d < q; d++) {
System.out.print(multiply[c][d] + "\t");
}
System.out.print("\n");
}
}
in.close(); // Close the scanner
}
}
```

Quadratic Probing Hash Table Errors and Fixes:

1.How many errors are there in the program? There is 1 error in the program.

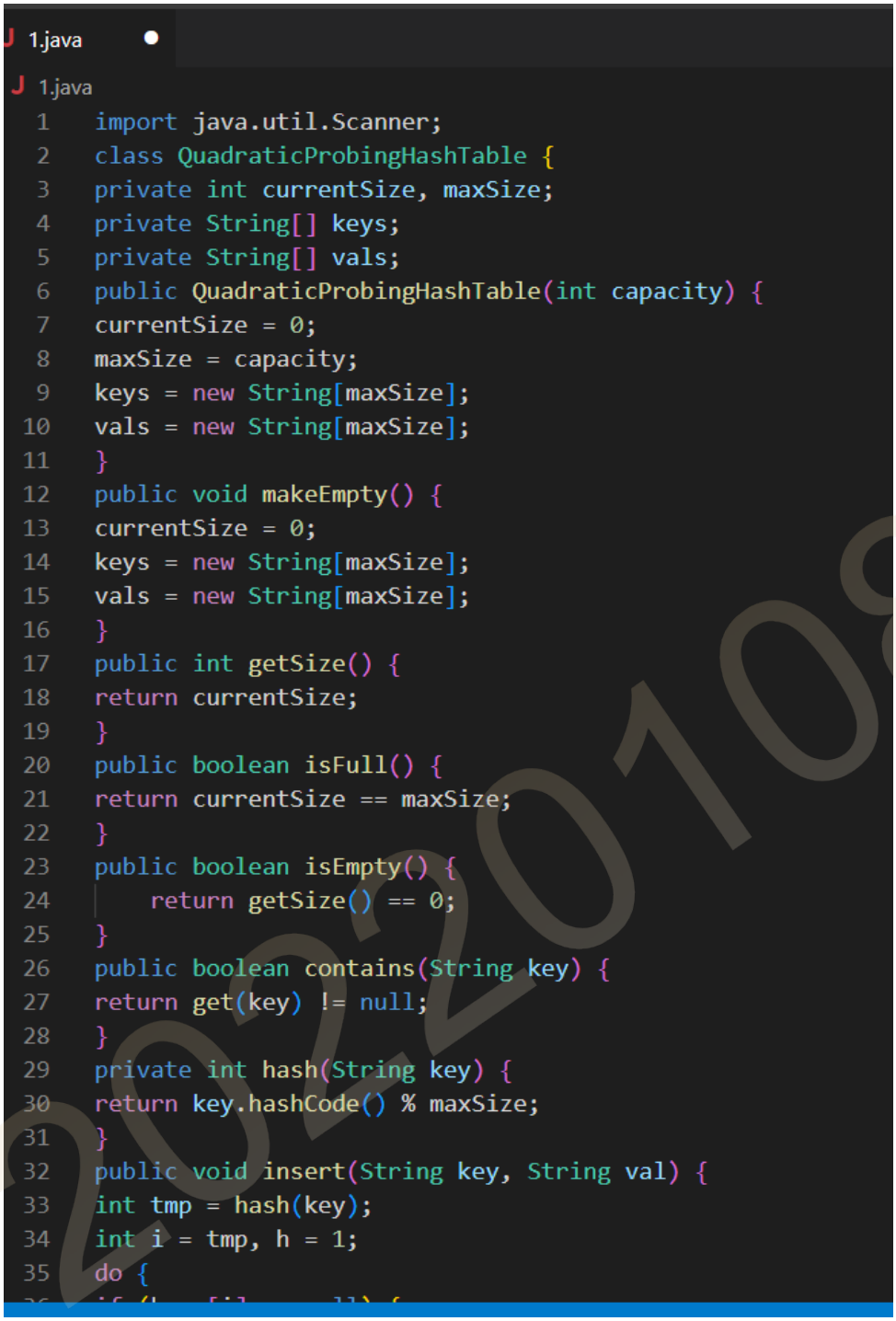
2. How many breakpoints do you need to fix this error? We need 1 breakpoint to fix this error.

Answer:

Steps Taken to Fix the Error:

Error: The line 'i += (i + h / h--) % maxSize;' in the insert method is incorrect.

Fix: Updated it to 'i = (i + h * h++) % maxSize;' to correctly implement quadratic probing.



1.java

1.java

```java
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
quadratic probing
} 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--;
    for (i = (i + h * h++) % maxSize; keys[i] !=
    null; i = (i + h * h++) % maxSize) {
    String tmp1 = keys[i], tmp2 = vals[i];
```

```java
1.java
1.java
68      null; i = (i + h * h++) % maxSize) {
69      String tmp1 = keys[i], tmp2 = vals[i];
70      keys[i] = vals[i] = null;
71      currentSize--;
72      insert(tmp1, tmp2);
73      }
74  }
75  public void printHashTable() {
76  System.out.println("\nHash Table:");
77  for (int i = 0; i < maxSize; i++)
78  if (keys[i] != null)
79  System.out.println(keys[i] + " " +
80  vals[i]);
81  System.out.println();
82  }
83  }
84  public class QuadraticProbingHashTableTest {
85  public static void main(String[] args) {
86  Scanner scan = new Scanner(System.in);
87  System.out.println("Hash Table Test\n\n");
88  System.out.println("Enter size");
89  QuadraticProbingHashTable qpht = new
90  QuadraticProbingHashTable(scan.nextInt());
91  char ch;
92  do {
93  System.out.println("\nHash Table
94  Operations\n");
95  System.out.println("1. insert ");
96  System.out.println("2. remove");
97  System.out.println("3. get");
98  System.out.println("4. clear");
99  System.out.println("5. size");
100 int choice = scan.nextInt();
101 switch (choice) {
102 case 1:
```

```java
1.java
1.java
103     System.out.println("Enter key and
104     value");
105     qpht.insert(scan.next(),
106     scan.next());
107     break;
108     case 2:
109     System.out.println("Enter key");
110     qpht.remove(scan.next());
111     break;
112     case 3:
113     System.out.println("Enter key");
114     System.out.println("Value = " +
115     qpht.get(scan.next()));
116     break;
117     case 4:
118     qpht.makeEmpty();
119     System.out.println("Hash Table
120     Cleared\n");
121     break;
122     case 5:
123     System.out.println("Size = " +
124     qpht.getSize());
125     break;
126     default:
127     System.out.println("Wrong Entry
128     \n");
129     break;
130     }
131     qpht.printHashTable();
132     System.out.println("\nDo you want to
133     continue (Type y or n) \n");
134     ch = scan.next().charAt(0);
135     } while (ch == 'Y' || ch == 'y');
136     }
137 }
```

Sorting Array Errors and Fixes:

1.How many errors are there in the program? There are 2 errors in the program.

2. How many breakpoints do you need to fix this error? We need 2 breakpoints to fix these errors.

Answer:

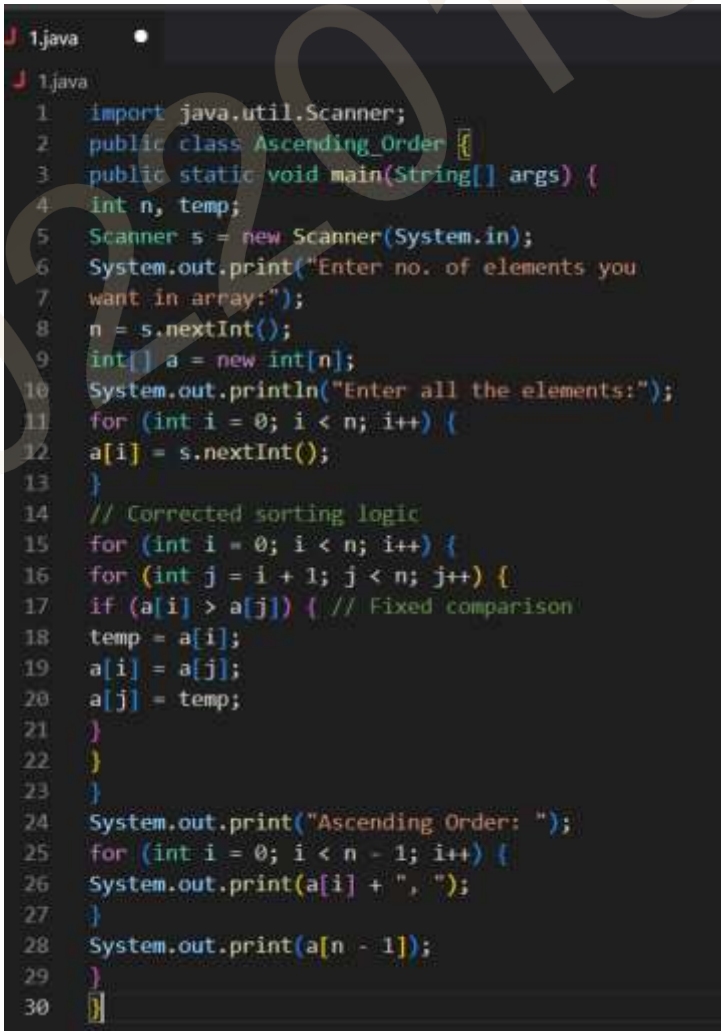
Steps Taken to Fix the Errors:

Error 1: The loop condition 'for (int i = 0; i >= n; i++);' is incorrect.

Fix 1: Changed it to 'for (int i = 0; i < n; i++)' to ensure proper iteration over the array.

Error 2: The condition in the inner loop 'if (a[i] <= a[j])' should be reversed.

Fix 2: Updated it to 'if (a[i] > a[j])' to correctly sort the array in ascending order.



Stack Implementation (from Stack Implementation.txt)

Errors and Fixes:

How many errors are there in the program?

There are 2 errors in the program.

How many breakpoints do you need to fix these errors?

We need 2 breakpoints to address these errors.

Answer:

Steps Taken to Fix the Errors:

Error 1: In the push method, the line `top--` is incorrect.

Fix 1: Changed it to `top++` to correctly increment the stack pointer.

Error 2: In the display method, the loop condition 'for (int i = 0; i > top; i++)' is incorrect.

Fix 2: Updated it to 'for (int i = 0; i <= top; i++)' to properly display all elements.

```java
1.java
1.java
27  public boolean isEmpty() {
28  return top == -1;
29  }
30  public void display() {
31  for (int i = 0; i <= top; i++) { // Corrected
32  loop condition
33  System.out.print(stack[i] + " ");
34  }
35  System.out.println();
36  }
37  }
38  public class StackReviseDemo {
39  public static void main(String[] args) {
40  StackMethods newStack = new StackMethods(5);
41  newStack.push(10);
42  newStack.push(1);
43  newStack.push(50);
44  newStack.push(20);
45  newStack.push(90);
46  newStack.display();
47  newStack.pop();
48  newStack.pop();
49  newStack.pop();
50  newStack.pop();
51  newStack.display();
52  }
53  }
54
```

Tower of Hanoi (from Tower of Hanoi.txt)(Tower of Hanoi)

Errors and Fixes:

1.How many errors are there in the program? There is 1 error in the program.

2. How many breakpoints do you need to fix this error? We need 1 breakpoint to fix this error.

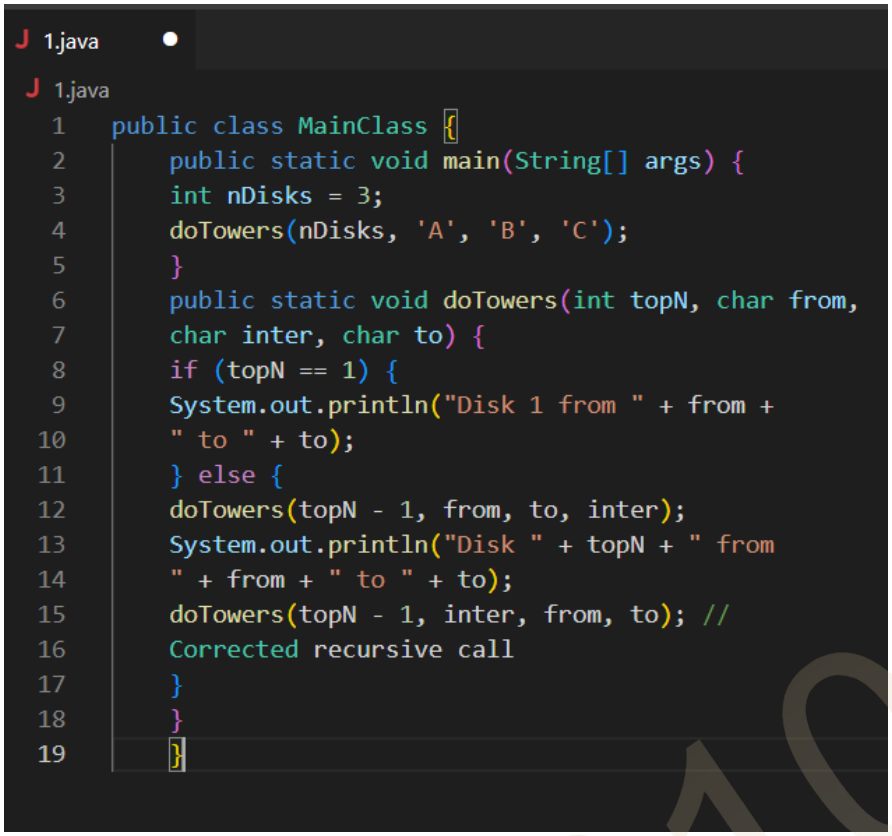
Answer:

Steps to Fix the Error:

Error: In the recursive call ‘doTowers(topN++, inter--, from + 1, to + 1);’, incorrect increments and decrements are being applied to the variables.

Fix: Change the call to ‘doTowers(topN - 1, inter, from, to);’ to ensure proper recursion and adherence to the Tower of Hanoi logic.

1.java



```java
public class MainClass {
    public static void main(String[] args) {
    int nDisks = 3;
    doTowers(nDisks, 'A', 'B', 'C');
    }
    public static void doTowers(int topN, char from,
    char inter, char to) {
    if (topN == 1) {
    System.out.println("Disk 1 from " + from +
    " to " + to);
    } else {
    doTowers(topN - 1, from, to, inter);
    System.out.println("Disk " + topN + " from
    " + from + " to " + to);
    doTowers(topN - 1, inter, from, to); //
    Corrected recursive call
    }
    }
}
```