IT-314

Software Engineering

LAB-07

Made by: Devarshi Patel

Student Id: 202201447

I. Program Inspection

An Error Checklist for Inspections

An important part of the inspection process is the use of a checklist to examine the program for common errors. Unfortunately, some checklists concentrate more on issues of style than on errors (for example, “Are comments accurate and meaningful?” and “Are if- else, code blocks, and do..while groups aligned?”), and the error checks are too nebulous to be useful (such as “Does the code meet the design requirements?”). The checklist in this section was compiled after many years of study of software errors. The checklist is largely language independent, meaning that most of the errors can occur with any programming language. You may wish to supplement this list with errors peculiar to your programming language and with errors detected after using the inspection process.

GitHub Code:

https://github.com/The-Young-Programmer/C-CPP-Programming/blob/main/Projects/C%20Projects/Advance%20(GUI)/Sci.%20Calculator%20(GUI)/main.c

Category A: Data Reference Errors

1. **Does a referenced variable have a value that is unset or uninitialized? This probably is the most frequent programming error; it occurs in a wide variety of circumstances. For each reference to a data item (variable, array element, field in a structure), attempt to “prove” informally that the item has a value at that point.**

   Yes

2. **For all array references, is each subscript value within the defined bounds of the corresponding dimension?**

   Yes

3. **For all array references, does each subscript have an integer value? This is not necessarily an error in all languages, but it is a dangerous practice.**

   Yes

4. **For all references through pointer or reference variables, is the referenced memory currently allocated? This is known as the “dangling reference” problem. It occurs in situations where the lifetime of a pointer is greater than the lifetime of the referenced memory. One situation occurs where a pointer references a local variable within a procedure, the pointer value is assigned to an output parameter or a global variable, the procedure returns (freeing the referenced location), and later the program attempts to use the pointer value. In a manner similar to**

checking for the prior errors, try to prove informally that, in each reference using a pointer variable, the reference memory exists.

No

5. **When a memory area has alias names with differing attributes, does the data value in this area have the correct attributes when referenced via one of these names? Situations to look for are the use of the EQUIVALENCE statement in FORTRAN, and the REDEFINES clause in COBOL. As an example, a FORTRAN program contains a real variable A and an integer variable B; both are made aliases for the same memory area by using an EQUIVALENCE statement. If the program stores a value into A and then references variable B, an error is likely present since the machine would use the floating-point bit representation in the memory area as an integer.**

   Not Applicable

6. **Does a variable's value have a type or attribute other than what the compiler expects? This situation might occur where a C, C++, or COBOL program reads a record into memory and references it by using a structure, but the physical representation of the record differs from the structure definition.**

   No

7. **Are there any explicit or implicit addressing problems if, on the machine being used, the units of memory allocation are smaller than the units of memory addressability? For instance, in some environments, fixed-length bit strings do not necessarily begin on byte boundaries, but addresses only point to byte boundaries. If a program computes the address of a bit string and later refers to the string through this address, the wrong memory location may be referenced.**

   This situation also could occur when passing a bit-string argument to a subroutine.

   No

8. **If pointer or reference variables are used, does the referenced memory location have the attributes the compiler expects? An example of such an error is where a C++pointer upon which a data structure is based is assigned the address of a different data structure.**

   Yes

9. **If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?**

   Yes

10. **When indexing into a string, are the limits of the string off by-one errors in indexing operations or in subscript references to arrays?**

No

11. **For object-oriented languages, are all inheritance requirements met in the implementing Class?**

    Not Applicable

Category B : Data-Declaration Errors

1. Have all variables been explicitly declared? A failure to do so is not necessarily an error, but it is a common source of trouble. For instance, if a program subroutine receives an array parameter, and fails to define the parameter as an array (as in a DIMENSION statement, for example), a reference to the array (such as C=A (I)) is interpreted as a function call, leading to the machine's attempting to execute the array as a program. Also, if a variable is not explicitly declared in an inner procedure or block, is it understood that the variable is shared with the enclosing block?

Yes

2. If all attributes of a variable are not explicitly stated in the declaration, are the defaults well understood? For instance, the default attributes received in Java are often a source of surprise.

Yes

3. Where a variable is initialized in a declarative statement, is it properly initialized? In many languages, initialization of arrays and strings is somewhat complicated and, hence, error prone.

No(Some variables are not initialized at the point of declaration)

4. Is each variable assigned the correct length and data type?

Yes

5. Is the initialization of a variable consistent with its memory type?

Yes

6. Are there any variables with similar names (VOLT and VOLTS, for example)? This is not necessarily an error, but it should be seen as a warning that the names may have been confused somewhere within the program.

No

Category C: Computation Error

1. Are there any computations using variables having inconsistent (such as non-arithmetic) data types?

No

2. Are there any mixed-mode computations? An example is the addition of a floating-point variable to an integer variable. Such occurrences are not necessarily errors, but they should be explored carefully to ensure that the language's conversion rules are understood. Consider the following Java snippet showing the rounding error that can occur when working with integers:

int x = 1;

int y = 2;

int z = 0;

z = x/y;

System.out.println ("z = " + z);

OUTPUT:

z = 0

Yes

3. Are there any computations using variables having the same data type but different lengths?

No

4. Is the data type of the target variable of an assignment smaller than the data type or result of the right-hand expression?

No

5. Is an overflow or underflow expression possible during the computation of an expression? That is, the end result may appear to have valid value, but an intermediate result might be too big or too small for the programming language's data types.

Yes

6. Is it possible for the divisor in a division operation to be zero?

Yes

7. If the underlying machine represents variables in base-2 form, are there any sequences of the resulting inaccuracy? That is, 10 x 0.1 is rarely equal to 1.0 on a binary machine.

Yes

8. Where applicable, can the value of a variable go outside the meaningful range? For example, statements assigning a value to the variable PROBABILITY might be checked to ensure that the assigned value will always be positive and not greater than

Yes

9. For expressions containing more than one operator, are the assumptions about the order of evaluation and precedence of operators correct?

Yes

10. Are there any invalid uses of integer arithmetic, particularly divisions? For instance, if i is an integer variable, whether the expression 2*i/2 == i depends on whether i has an odd or an even value and whether the multiplication or division is performed first.

No

Category D: Comparison Errors

1. Are there any comparisons between variables having different data types, such as comparing a character string to an address, date, or number?

No

2. Are there any mixed-mode comparisons or comparisons between variables of different lengths? If so, ensure that the conversion rules are well understood.

No

3. Are the comparison operators correct? Programmers frequently confuse such relations as at most, at least, greater than, not less than, less than or equal.

Yes

4. Does each Boolean expression state what it is supposed to state? Programmers often make mistakes when writing logical expressions involving and, or, and not.

Yes

5. Are the operands of a Boolean operator Boolean? Have comparison and Boolean operators been erroneously mixed together? This represents another frequent class of mistakes. Examples of a few typical mistakes are illustrated here. If you want to determine whether i is between 2 and 10, the expression 2<i<10 is incorrect; instead, it should be (2<i) && (i<10). If you want to determine whether i is greater than x or y, i>x||y is incorrect; instead, it should be (i>x)||(i>y). If you want to compare three numbers for

equality, if(a==b==c) does something quite different. If you want to test the mathematical relation x>y>z, the correct expression is(x>y)&&(y>z).

No

6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine? This is an occasional source of errors because of truncation and base-2 approximations of base-10 numbers.

Yes

7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct? That is, if you see an expression such as (if((a==2) && (b==2) || (c==3)), is it well understood whether the and or the or is performed first?

Yes

8. Does the way in which the compiler evaluates Boolean expressions affect the program? For instance, the statement if((x==0 && (x/y)>z) may be acceptable for compilers that end the test as soon as one side of an and is false, but may cause a division-by-zero error with other compilers.

No(No problematic Boolean expressions like if((x==0 && (x/y)>z) were found)

Category E: Control-Flow Errors

1. If the program contains a multiway branch such as a computed GO TO, can the index variable ever exceed the number of branch possibilities? For example, in the statement

GO TO (200, 300, 400), i

will i always have the value of 1, 2, or 3?

Not Applicable (No computed GO TO found in the provided code)

2. Will every loop eventually terminate? Devise an informal proof or argument showing that each loop will terminate.

Yes

3. Will the program, module, or subroutine eventually terminate?

Yes

4. Is it possible that, because of the conditions upon entry, a loop will never execute? If so, does this represent an over- sight? For instance, if you had the following loops headed by the following statements:

for (i==x ; i<=z; i++) {

...

}

while (NOTFOUND) {

...

}

what happens if NOTFOUND is initially false or if x is greater than z?

Yes

5. For a loop controlled by both iteration and a Boolean condition (a searching loop, for example) what are the consequences of loop fall-through? For example, for the psuedo-code loop headed by

DO I=1 to TABLESIZE WHILE (NOTFOUND)

what happens if NOTFOUND never becomes false?

Yes (Potential for infinite loop if the condition is never met)

6. Are there any off-by-one errors, such as one too many or too few iterations? This is a common error in zero-based loops. You will often forget to count "0" as a number. For example, if you want to create Java code for a loop that counted to 10, the following would be wrong, as it counts to 11:

for (int i=0; i<=10;i++) {

System.out.println(i);

}

Correct, the loop is iterated 10 times:

for (int i=0; i <=9;i++) {

System.out.println(i);

No

7. If the language contains a concept of statement groups or code blocks (e.g., do-while or {...}), is there an explicit while for each group and do the do's correspond to their

appropriate groups? Or is there a closing bracket for each open bracket? Most modern compilers will complain of such mismatches.

Yes

8. Are there any non-exhaustive decisions? For instance, if an input parameter's expected values are 1, 2, or 3, does the logic assume that it must be 3 if it is not 1 or 2? If so, is the assumption valid?

No

Category F: Interface Errors

1. Does the number of parameters received by this module equal the number of arguments sent by each of the calling modules? Also, is the order correct?

Yes

2. Do the attributes (e.g., data type and size) of each parameter match the attributes of each corresponding argument?

Yes

3. Does the units system of each parameter match the units system of each corresponding argument? For example, is the parameter expressed in degrees but the argument expressed in radians?

Yes

4. Does the number of arguments transmitted by this module to another module equal the number of parameters expected by that module?

Yes

5. Do the attributes of each argument transmitted to another module match the attributes of the corresponding parameter in that module?

Yes

6. Does the units system of each argument transmitted to another module match the units system of the corresponding parameter in that module?

Yes

7. If built-in functions are invoked, are the number, attributes, and order of the arguments correct?

Yes

8. Does a subroutine alter a parameter that is intended to be only an input value?

No

9. If global variables are present, do they have the same definition and attributes in all modules that reference them?

Yes

Category G: Input/Output Error

1. If files are explicitly declared, are their attributes correct?

No

2. Are the attributes on the file's OPEN statement correct?

No

3. Is there sufficient memory available to hold the file your program will read?

Yes

4. Have all files been opened before use?

No

5. Have all files been closed after use?

No

6. Are end-of-file conditions detected and handled correctly?

No

7. Are I/O error conditions handled correctly?

No

8. Are there spelling or grammatical errors in any text that is printed or displayed by the program?

No

Category H: Other Checks

1. **If the compiler produces a cross-reference listing of identifiers, examine it for variables that are never referenced or are referenced only once.**

   No

2. **If the compiler produces an attribute listing, check the attributes of each variable to ensure that no unexpected default attributes have been assigned.**

   No

3. **If the program compiled successfully, but the computer produced one or more "warning" or "informational" messages, check each one carefully. Warning messages are indications that the compiler suspects that you are doing something of questionable validity; all of these suspicions should be reviewed. Informational messages may list undeclared variables or language uses that impede code optimization.**

   No

4. **Is the program or module sufficiently robust? That is, does it check its input for validity?**

   No

5. **Is there a function missing from the program?**

No

Program Inspection:

1. **How many errors are there in the program? Mention the errors you have identified.**
    - **Use of non-standard headers: Headers like <conio.h>, <process.h>, <dos.h>,** and <graphics.h> are non-standard and may not be available on modern systems or compilers.
    - Potential missing function prototypes: Some functions like outtextxy and delay are used but not declared explicitly. These are likely from non-standard libraries and may cause errors if the correct libraries are not linked.
    - Unnecessary inclusions: Multiple header files are included that might not be necessary, such as both <iostream.h> and <stdio.h>.
    - Use of deprecated or non-standard functions: Functions like getch() from <conio.h> are outdated and might not work on many modern compilers.

    Total 4 errors identified so far.

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

    Code Walkthrough and Static Code Analysis would likely be the most effective since the issues relate to non-standard libraries and potential portability problems. Static analysis tools would flag deprecated or unsupported functions.

3. **Which type of error you are not able to identified using the program inspection?**
    - **Run-time Errors**: Issues like memory access violations, buffer overflows, or incorrect logic during execution are not easily detectable via static inspection.
    - **Performance Issues**: Problems related to inefficiency or high memory consumption may not be visible without running the program.
4. **Is the program inspection technique is worth applicable?**

    Yes, program inspection is worth applying. It helps catch portability issues, syntax errors, and improper function usage at an early stage, which can be crucial for ensuring the code is compatible with modern compilers and environments.

II. Code Debugging

1. Amstrong

Debugging

1. There is one error in the program related to the computation of the remainder, as previously identified.

2. To fix this error, one should set a breakpoint at the point where the remainder is computed to ensure it's calculated correctly. Step through the code to observe the values of variables and expressions during execution.

3. The corrected executable code is as follows:

```java
// Armstrong Number
class Armstrong {
    public static void main(String args[]) {
        int num = Integer.parseInt(args[0]);
        int n = num; // used to check at the last time
        int check = 0, remainder;
        while (num > 0) {
            remainder = num % 10;
            check = check + (int) Math.pow(remainder, 3);
            num = 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

Debugging

1. There are two errors in the program as mentioned above.

2. To fix these errors:

3. For Error 1 in the gcd function, you need one breakpoint at the beginning of the while loop to verify the correct execution of the loop.

4. For Error 2 in the lcm function, you would need to review the logic for calculating LCM, as it's a logical error.

5. The corrected executable code is as follows:

```java
import java.util.Scanner;

public class GCD_LCM {
    static int gcd(int x, int y) {
        int a, b;
        a = (x > y) ? x : y; // a is greater number
        b = (x < y) ? x : y; // b is smaller number
        while (b != 0) { // Fixed the while loop condition
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    static int lcm(int x, int y) {
        return (x * y) / gcd(x, y); // Calculate LCM using GCD
    }

    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

Debugging

1. There is one error in the program, as identified above.

2. To fix this error, you would need one breakpoint at the line: int option1 = opt[n][w]; to ensure n and w are correctly used without unintended increments.

3. The corrected executable code is as follows:

```
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);
        }

        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 the increment here
                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);
            }
        }
```

```
            // Rest of the code is fine

            // Print results
            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

Debugging

1. There are two errors in the program, as identified above.

2. To fix these errors, you would need one breakpoint at the beginning of the inner while loop to verify the execution of the loop. You can also use breakpoints to check the values of num and s during execution.

3. The corrected executable code is as follows:

```
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) { // Fixed the condition here
                s = s * (sum / 10);
                sum = sum % 10; // Fixed the missing semicolon
            }
            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

Debugging

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

2. To fix these errors, you would need to set breakpoints to examine the values of left, right, and array during execution. You can also use breakpoints to check the values of i1 and i2 inside the merge method.

3. The corrected executable code is as follows:

```
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 / 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. Multiply Matrices

Debugging

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

2. 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.

3. The corrected executable code is as follows:

```
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++) {
                for (k = 0; k < p; 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");
                }
            }
        }
    }
```

7. Quadratic Probing

Debugging

1. There are three errors in the program, as identified above.

2. To fix these errors, you would need to set breakpoints and step through the code while examining variables like i, h, tmp1, and tmp2. You should pay attention to the logic of the insert, remove, and get methods.

3. The corrected executable code is as follows:

```
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 = a String[maxSize];
    }

    public int getSize() {
        return currentSize;
    }
```

```java
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;
}
```

```java
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 += (h * h++) % maxSize;
    } 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;

        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.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

Debugging

1. There are two errors in the program as identified above.

2. To fix these errors, you need to set breakpoints and step through the code. You should focus on the class name, the loop conditions, and the unnecessary semicolon.

3. The corrected executable code is as follows:

```
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 the number of elements you want in the 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++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {
                    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

Debugging

1. There are three errors in the program, as identified above.

2. To fix these errors, you would need to set breakpoints and step through the code, focusing on the push, pop, and display methods. Correct the push and display

methods and add the missing pop method to provide a complete stack implementation.

3. The corrected executable code is as follows:

```
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++;
            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++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
```

10. Tower of Hanoi

Debugging

1. There is one error in the program, as identified above.

2. To fix this error, you need to replace the line:

doTowers(topN ++, inter--, from+1, to+1);

3. with the correct version:

doTowers(topN - 1, inter, from, to);

4. The corrected executable code is as follows:

```
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);
      }
   }
}
```

III. Static Analysis Tools

GitHub Code:

https://github.com/The-Young-Programmer/C-CPP-Programming/blob/main/Projects/C%20Projects/Advance%20(GUI)/Sci.%20Calculator%20(GUI)/main.c

We are analyzing the the code take in the Part I of the using the static analysis tool of c **CPPCheck .**

Results :

```
C:\Users\hitar>cppcheck --enable=all D:/main.c
Checking D:\main.c ...
D:\main.c:1:0: information: Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingI
ncludeSystem]
#include <stdio.h>
^
D:\main.c:2:0: information: Include file: <conio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingI
ncludeSystem]
#include <conio.h>
^
D:\main.c:3:0: information: Include file: <process.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missin
gIncludeSystem]
#include <process.h>
^
D:\main.c:4:0: information: Include file: <dos.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingInc
ludeSystem]
#include <dos.h>
^
D:\main.c:5:0: information: Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missing
IncludeSystem]
#include <stdlib.h>
^
D:\main.c:6:0: information: Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missi
ngIncludeSystem]
#include <iostream.h>
^
D:\main.c:7:0: information: Include file: <graphics.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missi
ngIncludeSystem]
#include <graphics.h>
^
D:\main.c:8:0: information: Include file: <math.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIn
cludeSystem]
#include <math.h>
^
D:\main.c:9:0: information: Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missing
IncludeSystem]
#include <string.h>
^
D:\main.c:0:0: information: Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches. [normalCheckLevelMaxBranches]

^
```

```
D:\main.c:797:38: warning: Opposite inner 'if' condition leads to a dead code block. [oppositeInnerCondition]
         if (ch == 'd' || ch == 'r' || ch == 'g') {
                                       ^
D:\main.c:796:16: note: outer condition: ch=='u'
        if (ch == 'u')
                  ^
D:\main.c:797:38: note: opposite inner condition: ch=='d'||ch=='r'||ch=='g'
         if (ch == 'd' || ch == 'r' || ch == 'g') {
                                       ^
D:\main.c:881:28: style: Condition 'ch1=='='' is always false [knownConditionTrueFalse]
      if (ch == '=' || ch1 == '=')
                           ^
D:\main.c:880:13: note: Assignment 'ch1='0'', assigned value is 48
      ch1 = '0';
            ^
D:\main.c:881:28: note: Condition 'ch1=='='' is always false
      if (ch == '=' || ch1 == '=')
                           ^
D:\main.c:402:29: style: The scope of the variable 'pnt1' can be reduced. [variableScope]
  double y = 0, z = 0, pnt, pnt1 = 0, x = 0, r = 0;
                            ^
D:\main.c:918:8: style: Local variable 'ch' shadows outer variable [shadowVariable]
  char ch;
       ^
D:\main.c:18:6: note: Shadowed declaration
char ch, ch1, ch2;
     ^
D:\main.c:918:8: note: Shadow variable
  char ch;
       ^
D:\main.c:32:45: style: Parameter 'string' can be declared as const array [constParameter]
void typeit(int x, int y, int spacing, char string[]) {
                                             ^
D:\main.c:43:13: style: Unused variable: h [unusedVariable]
  int x, y, h;
            ^
D:\main.c:402:34: style: Variable 'pnt1' is assigned a value that is never used. [unreadVariable]
  double y = 0, z = 0, pnt, pnt1 = 0, x = 0, r = 0;
                                 ^
D:\main.c:402:24: style: Unused variable: pnt [unusedVariable]
  double y = 0, z = 0, pnt, pnt1 = 0, x = 0, r = 0;
                       ^
```

```
                     ^
D:\main.c:404:32: style: Unused variable: errorcode [unusedVariable]
  int gdriver = DETECT, gmode, errorcode;
                               ^
D:\main.c:405:7: style: Unused variable: i [unusedVariable]
  int i;
      ^
D:\main.c:1048:9: style: Unused variable: a [unusedVariable]
    int a, b;
        ^
D:\main.c:1048:12: style: Unused variable: b [unusedVariable]
    int a, b;
           ^
nofile:0:0: information: Active checkers: 108/835 (use --checkers-report=<filename> to see details) [checkersReport]
```