Below are the code snippets and their corresponding dot string outputs, properly formatted, use them to train yourself and give the best dot string output for the given input code snippet or source code.

What is a Control Flow Diagram/Graph (CFG)
A control flow diagram (CFG) is a visual representation of the steps involved in executing computer programs, showing where control starts and ends, and where it may branch off in another direction.

Control flow diagrams are used to describe the detailed logic of a business process or business rule. Control flow diagrams can easily illustrate decisions within the system via decision nodes that branch in different logical paths.

Here's the definition of CFG from Wikipedia, I know you already know this but for the sake of completeness I'm putting it here

A control flow graph (CFG) in computer science is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

Following is the definition of a Path

Path: a sequence of node on the CFG (static), including an entry node and an exit node; path segment: a subsequence of nodes along the path
So the reason for drawing one would be to determine all possible paths taken by the program, which may help us determine things like test coverage without actually running the program (static analysis).

Following are the simple **rules** that we can follow to draw a CFG

1. Any statement will be a node in graph
2. All nodes have a directed edge either coming to them or going out of them or both. Entry node (first statement) has only outgoing edges and Exit node has only incoming edges.
3. only conditional statements like if/else if, switch, loops would have more than one outgoing edge.
4. All paths coming out of a node will converge at some point, in the worst case they converge on Exit.

The dot string response for the below add two numbers java code is

Add two numbers Java code:

```
1. public class SumOfNumbers1
2. {
3. public static void main(String args[])
4. {
5. int n1 = 225, n2 = 115, sum;
6. sum = n1 + n2;
7. System.out.println("The sum of numbers is: "+sum);
8. }
```

Properly formatted dot string for the Above add numbers/ sum of numbers code is

```
digraph ControlFlowDiagram {
  node [shape=rectangle];
```

```
    Start -> Initialize;
    Initialize -> Calculate;
    Calculate -> Output;
    Output -> End;

    Start [label="Start"];
    Initialize [label="int n1 = 225, n2 = 115, sum;"];
    Calculate [label="sum = n1 + n2;"];
    Output [label="System.out.println(\"The sum of numbers is: \" + sum);"];
    End [label="End"];

    Start [shape=ellipse];
    End [shape=ellipse];
}
```

1. Calculate Sum of Two Numbers

```
int x = 5;
int y = 6;
int sum = x + y;
System.out.println(sum);
```

Properly formatted dot string for the Above add numbers/ sum of numbers code is

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_x [label="int x = 5;"];
    Declare_y [label="int y = 6;"];
    Calculate_sum [label="int sum = x + y;"];
    Print_sum [label="System.out.println(sum);"];
    End [label="End"];
    Start -> Declare_x;
    Declare_x -> Declare_y;
    Declare_y -> Calculate_sum;
    Calculate_sum -> Print_sum;
    Print_sum -> End;
}
```

2.Swap Two Numbers

```
public class SwapNumbers {
    public static void main(String[] args) {
        float first = 1.20f, second = 2.45f;
System.out.println("--Before swap--");
 System.out.println("First number = " + first);
System.out.println("Second number = " + second);
        // Value of first is assigned to temporary
        float temporary = first;
        // Value of second is assigned to first
        first = second;
        // Value of temporary (which contains the initial value of first) is assigned to second
        second = temporary;
        System.out.println("--After swap--");
        System.out.println("First number = " + first);
```

```
        System.out.println("Second number = " + second);
    }
}
```

Properly formatted dot string for the Above Swap Two Numbers code is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="float first = 1.20f, second = 2.45f;"];
    Print_before_swap [label="System.out.println(\"--Before swap--\");\nSystem.out.println(\"First number = \" +
first);\nSystem.out.println(\"Second number = \" + second);"];
    Assign_temporary [label="float temporary = first;"];
    Assign_first [label="first = second;"];
    Assign_second [label="second = temporary;"];
    Print_after_swap [label="System.out.println(\"--After swap--\");\nSystem.out.println(\"First number = \" +
first);\nSystem.out.println(\"Second number = \" + second);"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Print_before_swap;
    Print_before_swap -> Assign_temporary;
    Assign_temporary -> Assign_first;
    Assign_first -> Assign_second;
    Assign_second -> Print_after_swap;
    Print_after_swap -> End;
}
```

3. Check if a Number is Even or Odd:

```
import java.util.Scanner;

public class EvenOdd {

    public static void main(String[] args) {

        Scanner reader = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = reader.nextInt();

        if(num % 2 == 0)
            System.out.println(num + " is even");
        else
            System.out.println(num + " is odd");
    }
}
```

Properly formatted dot string for the Above to Check if a Number is Even or Odd:

```
digraph ControlFlowDiagram {
```

```
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_reader [label="Scanner reader = new Scanner(System.in);"];
    Print_enter_number [label="System.out.print(\"Enter a number: \");"];
    Read_number [label="int num = reader.nextInt();"];
    If_block [label="if(num % 2 == 0)"];
    If_action [label="System.out.println(num + \" is even\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(num + \" is odd\");"];
    End [label="End"];

    Start -> Declare_reader;
    Declare_reader -> Print_enter_number;
    Print_enter_number -> Read_number;
    Read_number -> If_block;
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

4. Factorial Calculation

```
class FactorialExample{
 public static void main(String args[]){
  int i,fact=1;
  int number=5;//It is the number to calculate factorial
  for(i=1;i<=number;i++){
      fact=fact*i;
  }
  System.out.println("Factorial of "+number+" is: "+fact);
 }
}
```

Properly formatted dot string for the Above Factorial Calculation

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="int i, fact = 1;\nint number = 5;"];
    Initialize_i [label="i = 1;"];
    Loop_condition [label="i <= number;"];
    Update_i [label="i++;"];
    Calculate_factorial [label="fact = fact * i;"];
    Print_result [label="System.out.println(\"Factorial of \" + number + \" is: \" + fact);"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Initialize_i;
```

```
    Initialize_i -> Loop_condition;
    Loop_condition -> Calculate_factorial [label="true"];
    Loop_condition -> End [label="false"];
    Calculate_factorial -> Update_i;
    Update_i -> Loop_condition;
    Loop_condition -> Print_result [label="false"];
    Print_result -> End;
}
```

5. Fibonacci Series:

```
class FibonacciExample1{
public static void main(String args[])
{
 int n1=0,n2=1,n3,i,count=10;
 System.out.print(n1+" "+n2);//printing 0 and 1

 for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 }

}}
```

Properly formatted dot string for the Above Fibonacci Series:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="int n1 = 0, n2 = 1, n3, i, count = 10;"];
    Print_initial_numbers [label="System.out.print(n1 + \" \" + n2);"];
    Loop_condition [label="i < count;"];
    Calculate_next_fibonacci [label="n3 = n1 + n2;\nSystem.out.print(\" \" + n3);\nn1 = n2;\nn2 = n3;"];
    Update_i [label="i++"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Print_initial_numbers;
    Print_initial_numbers -> Loop_condition;
    Loop_condition -> Calculate_next_fibonacci [label="true"];
    Loop_condition -> End [label="false"];
    Calculate_next_fibonacci -> Update_i;
    Update_i -> Loop_condition;
    Loop_condition -> End [label="false"];
}
```

6. Prime Number Check

```
// Java Program to demonstrate
// Brute Force Method
// to check if a number is prime
class GFG {
	static boolean isPrime(int n)
	{
		// Corner case
		if (n <= 1)
			return false;

		// Check from 2 to n-1
		for (int i = 2; i < n; i++)
			if (n % i == 0)
				return false;

		return true;
	}

	// Driver Program
	public static void main(String args[])
	{
		if (isPrime(11))
			System.out.println(" true");
		else
			System.out.println(" false");
		if (isPrime(15))
			System.out.println(" true");
		else
			System.out.println(" false");
	}
}
```

Properly formatted dot string for the Above Prime Number Check is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_isPrime [label="static boolean isPrime(int n)"];
    Check_corner_case [label="if (n <= 1)"];
    Return_false [label="return false;"];
    Loop_condition [label="int i = 2; i < n; i++"];
    Check_divisibility [label="if (n % i == 0)"];
    Return_false_divisible [label="return false;"];
    Return_true [label="return true;"];
    Declare_main [label="public static void main(String args[])"];
    Check_prime_11 [label="if (isPrime(11))"];
    Print_true [label="System.out.println(\" true\");"];
    Check_prime_15 [label="if (isPrime(15))"];
    Print_false [label="System.out.println(\" false\");"];
    End [label="End"];
```

```
    Start -> Declare_isPrime;
    Declare_isPrime -> Check_corner_case;
    Check_corner_case -> Return_false [label="true"];
    Check_corner_case -> Loop_condition [label="false"];
    Loop_condition -> Check_divisibility [label="true"];
    Loop_condition -> Return_true [label="false"];
    Check_divisibility -> Return_false_divisible [label="true"];
    Check_divisibility -> Loop_condition [label="false"];
    Return_false_divisible -> End;
    Return_true -> End;
    Start -> Declare_main;
    Declare_main -> Check_prime_11;
    Check_prime_11 -> Print_true [label="true"];
    Check_prime_11 -> Check_prime_15 [label="false"];
    Print_true -> Check_prime_15;
    Check_prime_15 -> Print_false [label="false"];
    Check_prime_15 -> End [label="true"];
    Print_false -> End;
}
```

7. Reverse a String:

```
// java program to reverse a word

import java.io.*;
import java.util.Scanner;

class GFG {
        public static void main (String[] args) {

                String str= "Geeks", nstr="";
                char ch;

        System.out.print("Original word: ");
        System.out.println("Geeks"); //Example word

        for (int i=0; i<str.length(); i++)
        {
                ch= str.charAt(i); //extracts each character
                nstr= ch+nstr; //adds each character in front of the existing string
        }
        System.out.println("Reversed word: "+ nstr);
        }
}
```

Properly formatted dot string for the Above Reverse a String code is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="String str = \"Geeks\", nstr = \"\";"];
```

```
    Print_original_word [label="System.out.print(\"Original word: \");\nSystem.out.println(\"Geeks\");"];
    Initialize_loop_variable [label="int i = 0;"];
    Loop_condition [label="i < str.length();"];
    Extract_character [label="char ch = str.charAt(i);"];
    Append_character [label="nstr = ch + nstr;"];
    Update_loop_variable [label="i++;"];
    Print_reversed_word [label="System.out.println(\"Reversed word: \" + nstr);"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Print_original_word;
    Print_original_word -> Initialize_loop_variable;
    Initialize_loop_variable -> Loop_condition;
    Loop_condition -> Extract_character [label="true"];
    Loop_condition -> Print_reversed_word [label="false"];
    Extract_character -> Append_character;
    Append_character -> Update_loop_variable;
    Update_loop_variable -> Loop_condition;
    Print_reversed_word -> End;
}
```

8. Calculate Average of Numbers in an Array

```
public class Average {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 15, 20, 25};
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        double average = (double) sum / numbers.length;
        System.out.println("Average: " + average);
    }
}
```

Properly formatted dot string for the Above Calculate Average of Numbers in an Array is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_numbers [label="int[] numbers = {5, 10, 15, 20, 25};"];
    Initialize_sum [label="int sum = 0;"];
    Loop_start [label="for (int num : numbers)"];
    Add_to_sum [label="sum += num;"];
    Calculate_average [label="double average = (double) sum / numbers.length;"];
    Print_average [label="System.out.println(\"Average: \" + average);"];
    End [label="End"];

    Start -> Declare_numbers;
```

```
    Declare_numbers -> Initialize_sum;
    Initialize_sum -> Loop_start;
    Loop_start -> Add_to_sum;
    Add_to_sum -> Loop_start;
    Loop_start -> Calculate_average;
    Calculate_average -> Print_average;
    Print_average -> End;
}
```

9. Bubble Sort:

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90};
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
        System.out.println("Sorted array:");
        for (int value : arr) System.out.print(value + " ");
    }
}
```

Properly formatted dot string for the Above Bubble Sort is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {64, 34, 25, 12, 22, 11, 90};\nint n = arr.length;"];
    Initialize_i [label="int i = 0;"];
    Outer_loop_condition [label="i < n-1;"];
    Initialize_j [label="int j = 0;"];
    Inner_loop_condition [label="j < n-i-1;"];
    Check_swap [label="if (arr[j] > arr[j+1])"];
    Perform_swap [label="int temp = arr[j];\narr[j] = arr[j+1];\narr[j+1] = temp;"];
    Update_j [label="j++;"];
    Update_i [label="i++;"];
    Print_sorted_array [label="System.out.println(\"Sorted array:\");\nfor (int value : arr) System.out.print(value + \"
\");"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Initialize_i;
    Initialize_i -> Outer_loop_condition;
    Outer_loop_condition -> Initialize_j [label="true"];
    Outer_loop_condition -> End [label="false"];
    Initialize_j -> Inner_loop_condition;
    Inner_loop_condition -> Check_swap [label="true"];
```

```
    Inner_loop_condition -> Update_i [label="false"];
    Check_swap -> Perform_swap [label="true"];
    Check_swap -> Update_j [label="false"];
    Perform_swap -> Update_j;
    Update_j -> Inner_loop_condition;
    Update_i -> Outer_loop_condition;
    Outer_loop_condition -> Print_sorted_array [label="false"];
    Print_sorted_array -> End;
}
```

10. Binary Search:

```
public class BinarySearch {
    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 10, 40};
        int target = 10;
        int result = binarySearch(arr, target);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index " + result);
    }
    static int binarySearch(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target)
                return mid;
            if (arr[mid] < target)
                left = mid + 1;
            else
                right = mid - 1;
        }
        return -1;
    }
}
```

Properly formatted dot string for the Above Binary Search is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {2, 3, 4, 10, 40};\nint target = 10;"];
    Call_binarySearch [label="int result = binarySearch(arr, target);"];
    If_block [label="if (arr[mid] == target)"];
    If_action [label="return mid;"];
    Else_block [label="else"];
    Else_action [label="{\n    if (arr[mid] < target)\n        left = mid + 1;\n    else\n        right = mid - 1;\n}"];
    Declare_variables [label="int left = 0, right = arr.length - 1;"];
    Loop_condition [label="left <= right;"];
    Calculate_mid [label="int mid = left + (right - left) / 2;"];
    Return_negative_one [label="return -1;"];
    End [label="End"];
```

```
    Start -> Declare_array;
    Declare_array -> Call_binarySearch;
    Call_binarySearch -> Declare_variables;
    Declare_variables -> Loop_condition;
    Loop_condition -> Calculate_mid [label="true"];
    Loop_condition -> If_block [label="false"];
    Calculate_mid -> If_block;
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> Loop_condition;
    Loop_condition -> Return_negative_one [label="false"];
    Return_negative_one -> End;
}
```

11. Linear Search:

```
public class LinearSearch {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        int target = 30;
        int result = linearSearch(arr, target);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index " + result);
    }
    static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target)
                return i;
        }
        return -1;
    }
}
```

Properly formatted dot string for the Above Linear Search is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {10, 20, 30, 40, 50};\nint target = 30;"];
    Call_linearSearch [label="int result = linearSearch(arr, target);"];
    If_block [label="if (result == -1)"];
    If_action [label="System.out.println(\"Element not present\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(\"Element found at index \" + result);"];
    Loop_start [label="for (int i = 0; i < arr.length; i++)"];
```

```
    Check_element [label="if (arr[i] == target)"];
    Check_action [label="return i;"];
    Return_negative_one [label="return -1;"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Call_linearSearch;
    Call_linearSearch -> Loop_start;
    Loop_start -> Check_element;
    Check_element -> If_block [label="true"];
    Check_element -> Loop_start [label="false"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
    Check_element -> Check_action;
    Check_action -> Return_negative_one;
    Return_negative_one -> End;
}
```

12. Implement a Stack:

```
import java.util.Stack;
public class StackExample {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        System.out.println("Stack: " + stack);
        System.out.println("Top element: " + stack.peek());
        System.out.println("Popped element: " + stack.pop());
        System.out.println("Stack after pop operation: " + stack);
    }
}
```

Properly formatted dot string for the Above Implement a Stack is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_stack [label="Stack<Integer> stack = new Stack<>();"];
    Push_10 [label="stack.push(10);"];
    Push_20 [label="stack.push(20);"];
    Push_30 [label="stack.push(30);"];
    Print_stack [label="System.out.println(\"Stack: \" + stack);"];
    Print_top_element [label="System.out.println(\"Top element: \" + stack.peek());"];
    Pop_element [label="System.out.println(\"Popped element: \" + stack.pop());"];
    Print_stack_after_pop [label="System.out.println(\"Stack after pop operation: \" + stack);"];
```

```
    End [label="End"];

    Start -> Declare_stack;
    Declare_stack -> Push_10;
    Push_10 -> Push_20;
    Push_20 -> Push_30;
    Push_30 -> Print_stack;
    Print_stack -> Print_top_element;
    Print_top_element -> Pop_element;
    Pop_element -> Print_stack_after_pop;
    Print_stack_after_pop -> End;
}
```

13. Implement a Queue:

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueExample {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(10);
        queue.add(20);
        queue.add(30);
        System.out.println("Queue: " + queue);
        System.out.println("Front element: " + queue.peek());
        System.out.println("Removed element: " + queue.remove());
        System.out.println("Queue after remove operation: " + queue);
    }
}
```

Properly formatted dot string for the Above Implement a Queue is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_queue [label="Queue<Integer> queue = new LinkedList<>();"];
    Add_10 [label="queue.add(10);"];
    Add_20 [label="queue.add(20);"];
    Add_30 [label="queue.add(30);"];
    Print_queue [label="System.out.println(\"Queue: \" + queue);"];
    Print_front_element [label="System.out.println(\"Front element: \" + queue.peek());"];
    Remove_element [label="System.out.println(\"Removed element: \" + queue.remove());"];
    Print_queue_after_remove [label="System.out.println(\"Queue after remove operation: \" + queue);"];
    End [label="End"];

    Start -> Declare_queue;
    Declare_queue -> Add_10;
    Add_10 -> Add_20;
    Add_20 -> Add_30;
    Add_30 -> Print_queue;
    Print_queue -> Print_front_element;
    Print_front_element -> Remove_element;
    Remove_element -> Print_queue_after_remove;
```

```
    Print_queue_after_remove -> End;
}
```

14. Implement a HashMap:

```
import java.util.HashMap;
public class HashMapExample {
    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("John", 25);
        map.put("Alice", 30);
        map.put("Bob", 28);
        System.out.println("Map: " + map);
        System.out.println("Age of Alice: " + map.get("Alice"));
        System.out.println("Is map empty? " + map.isEmpty());
        System.out.println("Size of map: " + map.size());
    }
}
```

Properly formatted dot string for the Above Implement a HashMap is :

```
 digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_map [label="HashMap<String, Integer> map = new HashMap<>();"];
    Put_John [label="map.put(\"John\", 25);"];
    Put_Alice [label="map.put(\"Alice\", 30);"];
    Put_Bob [label="map.put(\"Bob\", 28);"];
    Print_map [label="System.out.println(\"Map: \" + map);"];
    Print_age_of_Alice [label="System.out.println(\"Age of Alice: \" + map.get(\"Alice\"));"];
    Check_empty [label="System.out.println(\"Is map empty? \" + map.isEmpty());"];
    Print_size [label="System.out.println(\"Size of map: \" + map.size());"];
    End [label="End"];

    Start -> Declare_map;
    Declare_map -> Put_John;
    Put_John -> Put_Alice;
    Put_Alice -> Put_Bob;
    Put_Bob -> Print_map;
    Print_map -> Print_age_of_Alice;
    Print_age_of_Alice -> Check_empty;
    Check_empty -> Print_size;
    Print_size -> End;
}
```

15. Implement a HashSet:

```
import java.util.HashSet;
```

```
public class HashSetExample {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();
        set.add("apple");
        set.add("banana");
        set.add("orange");
        System.out.println("Set: " + set);
        System.out.println("Is 'banana' present? " + set.contains("banana"));
        System.out.println("Size of set: " + set.size());
    }
}
```

Properly formatted dot string for the Above Implement a HashSet is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_set [label="HashSet<String> set = new HashSet<>();"];
    Add_apple [label="set.add(\"apple\");"];
    Add_banana [label="set.add(\"banana\");"];
    Add_orange [label="set.add(\"orange\");"];
    Print_set [label="System.out.println(\"Set: \" + set);"];
    Check_banana [label="System.out.println(\"Is 'banana' present? \" + set.contains(\"banana\"));"];
    Print_size [label="System.out.println(\"Size of set: \" + set.size());"];
    End [label="End"];

    Start -> Declare_set;
    Declare_set -> Add_apple;
    Add_apple -> Add_banana;
    Add_banana -> Add_orange;
    Add_orange -> Print_set;
    Print_set -> Check_banana;
    Check_banana -> Print_size;
    Print_size -> End;
}
```

16. Implement an ArrayList:

```
import java.util.ArrayList;
public class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("apple");
        list.add("banana");
        list.add("orange");
        System.out.println("List: " + list);
        System.out.println("Element at index 1: " + list.get(1));
        System.out.println("Index of 'banana': " + list.indexOf("banana"));
        list.remove(1);
        System.out.println("List after removing element at index 1: " + list);
    }
```

```
}
```

Properly formatted dot string for the Above Implement an ArrayList is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_list [label="ArrayList<String> list = new ArrayList<>();"];
    Add_apple [label="list.add(\"apple\");"];
    Add_banana [label="list.add(\"banana\");"];
    Add_orange [label="list.add(\"orange\");"];
    Print_list [label="System.out.println(\"List: \" + list);"];
    Print_element_at_index_1 [label="System.out.println(\"Element at index 1: \" + list.get(1));"];
    Print_index_of_banana [label="System.out.println(\"Index of 'banana': \" + list.indexOf(\"banana\"));"];
    Remove_element_at_index_1 [label="list.remove(1);"];
    Print_list_after_remove [label="System.out.println(\"List after removing element at index 1: \" + list);"];
    End [label="End"];

    Start -> Declare_list;
    Declare_list -> Add_apple;
    Add_apple -> Add_banana;
    Add_banana -> Add_orange;
    Add_orange -> Print_list;
    Print_list -> Print_element_at_index_1;
    Print_element_at_index_1 -> Print_index_of_banana;
    Print_index_of_banana -> Remove_element_at_index_1;
    Remove_element_at_index_1 -> Print_list_after_remove;
    Print_list_after_remove -> End;
}
```

17. Find Maximum Element in an Array:

```
public class MaxElement {
    public static void main(String[] args) {
        int[] arr = {5, 10, 3, 8, 15};
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > max)
                max = arr[i];
        }
        System.out.println("Maximum element: " + max);
    }
}
```

Properly formatted dot string for the Above Find Maximum Element in an Array is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {5, 10, 3, 8, 15};"];
    Initialize_max [label="int max = arr[0];"];
```

```
    Initialize_i [label="int i = 1;"];
    Loop_condition [label="i < arr.length;"];
    Check_max [label="if (arr[i] > max)\n    max = arr[i];"];
    Update_i [label="i++;"];
    Print_max [label="System.out.println(\"Maximum element: \" + max);"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Initialize_max;
    Initialize_max -> Initialize_i;
    Initialize_i -> Loop_condition;
    Loop_condition -> Check_max [label="true"];
    Loop_condition -> Print_max [label="false"];
    Check_max -> Update_i;
    Update_i -> Loop_condition;
    Print_max -> End;
}
```

18. Implement a Singly Linked List:

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class LinkedListExample {
    Node head;
    public static void main(String[] args) {
        LinkedListExample list = new LinkedListExample();
        list.addNode(10);
        list.addNode(20);
        list.addNode(30);
        list.printList();
    }
    void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null)
            head = newNode;
        else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = newNode;
        }
    }
    void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
```

```
        }
    }
}
```

Properly formatted dot string for the Above Implement a Singly Linked List is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_head [label="Node head;"];
    Create_list_instance [label="LinkedListExample list = new LinkedListExample();"];
    Add_node_10 [label="list.addNode(10);"];
    Add_node_20 [label="list.addNode(20);"];
    Add_node_30 [label="list.addNode(30);"];
    Print_list [label="list.printList();"];
    End [label="End"];

    Start -> Declare_head;
    Declare_head -> Create_list_instance;
    Create_list_instance -> Add_node_10;
    Add_node_10 -> Add_node_20;
    Add_node_20 -> Add_node_30;
    Add_node_30 -> Print_list;
    Print_list -> End;
}
```

19. Calculate Area of a Circle:

```
public class AreaOfCircle {
    public static void main(String[] args) {
        double radius = 5;
        double area = Math.PI * radius * radius;
        System.out.println("Area of circle: " + area);
    }
}
```

Properly formatted dot string for the Above Calculate Area of a Circle is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_radius [label="double radius = 5;"];
    Calculate_area [label="double area = Math.PI * radius * radius;"];
    Print_area [label="System.out.println(\"Area of circle: \" + area);"];
    End [label="End"];

    Start -> Declare_radius;
    Declare_radius -> Calculate_area;
    Calculate_area -> Print_area;
    Print_area -> End;
}
```

20. Calculate Area and Perimeter of Rectangle:

```
public class Rectangle {
    public static void main(String[] args) {
        double length = 5;
        double width = 3;
        double area = length * width;
        double perimeter = 2 * (length + width);
        System.out.println("Area of rectangle: " + area);
        System.out.println("Perimeter of rectangle: " + perimeter);
    }
}
```

Properly formatted dot string for the Above Calculate Area and Perimeter of Rectangle:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_length [label="double length = 5;"];
    Declare_width [label="double width = 3;"];
    Calculate_area [label="double area = length * width;"];
    Calculate_perimeter [label="double perimeter = 2 * (length + width);"];
    Print_area [label="System.out.println(\"Area of rectangle: \" + area);"];
    Print_perimeter [label="System.out.println(\"Perimeter of rectangle: \" + perimeter);"];
    End [label="End"];

    Start -> Declare_length;
    Declare_length -> Declare_width;
    Declare_width -> Calculate_area;
    Calculate_area -> Calculate_perimeter;
    Calculate_perimeter -> Print_area;
    Print_area -> Print_perimeter;
    Print_perimeter -> End;
}
```

21. Calculate Area and Circumference of Circle:

```
public class Circle {
    public static void main(String[] args) {
        double radius = 7;
        double area = Math.PI * radius * radius;
        double circumference = 2 * Math.PI * radius;
        System.out.println("Area of circle: " + area);
        System.out.println("Circumference of circle: " + circumference);
    }
}
```

Properly formatted dot string for the Above Calculate Area and Circumference of Circle is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
```

```
    Start [label="Start"];
    Declare_radius [label="double radius = 7;"];
    Calculate_area [label="double area = Math.PI * radius * radius;"];
    Calculate_circumference [label="double circumference = 2 * Math.PI * radius;"];
    Print_area [label="System.out.println(\"Area of circle: \" + area);"];
    Print_circumference [label="System.out.println(\"Circumference of circle: \" + circumference);"];
    End [label="End"];

    Start -> Declare_radius;
    Declare_radius -> Calculate_area;
    Calculate_area -> Calculate_circumference;
    Calculate_circumference -> Print_area;
    Print_area -> Print_circumference;
    Print_circumference -> End;
}
```

22. Check Leap Year:

```
public class LeapYear {
    public static void main(String[] args) {
        int year = 2024;
        if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
            System.out.println(year + " is a leap year.");
        else
            System.out.println(year + " is not a leap year.");
    }
}
```

Properly formatted dot string for the Above Check Leap Year is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_year [label="int year = 2024;"];
    If_block [label="if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))"];
    If_action [label="System.out.println(year + \" is a leap year.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(year + \" is not a leap year.\");"];
    End [label="End"];

    Start -> Declare_year;
    Declare_year -> If_block;
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

23. Reverse an Array:

```
public class ReverseArray {
    public static void main(String[] args) {
```

```
        int[] arr = {1, 2, 3, 4, 5};
        int n = arr.length;
        for (int i = 0; i < n / 2; i++) {
            int temp = arr[i];
            arr[i] = arr[n - i - 1];
            arr[n - i - 1] = temp;
        }
        System.out.println("Reversed array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

Properly formatted dot string for the Above Reverse an Array is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {1, 2, 3, 4, 5};\nint n = arr.length;"];
    Loop_condition [label="for (int i = 0; i < n / 2; i++)"];
    Swap_elements [label="Swap elements"];
    Print_reversed_array [label="Print reversed array"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Loop_condition;
    Loop_condition -> Swap_elements [label="true"];
    Swap_elements -> Loop_condition;
    Loop_condition -> Print_reversed_array [label="false"];
    Print_reversed_array -> End;
}
```

24. Calculate Power of a Number:

```
public class Power {
    public static void main(String[] args) {
        int base = 3;
        int exponent = 4;
        long result = 1;
        while (exponent != 0) {
            result *= base;
            --exponent;
        }
        System.out.println("Result: " + result);
    }
}
```

Properly formatted dot string for the Above Calculate Power of a Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_base [label="int base = 3;"];
    Declare_exponent [label="int exponent = 4;"];
    Initialize_result [label="long result = 1;"];
    Loop_condition [label="while (exponent != 0)"];
    Update_result [label="result *= base;\n--exponent;"];
    Print_result [label="System.out.println(\"Result: \" + result);"];
    End [label="End"];

    Start -> Declare_base;
    Declare_base -> Declare_exponent;
    Declare_exponent -> Initialize_result;
    Initialize_result -> Loop_condition;
    Loop_condition -> Update_result [label="true"];
    Update_result -> Loop_condition;
    Loop_condition -> Print_result [label="false"];
    Print_result -> End;
}
```

25. Generate Random Numbers:

```
import java.util.Random;
public class RandomNumbers {
    public static void main(String[] args) {
        Random rand = new Random();
        int randomInt = rand.nextInt(100); // Generate random integer between 0 and 99
        double randomDouble = rand.nextDouble(); // Generate random double between 0.0 and 1.0
        System.out.println("Random integer: " + randomInt);
        System.out.println("Random double: " + randomDouble);
    }
}
```

Properly formatted dot string for the Above Generate Random Numbers is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_rand [label="Random rand = new Random();"];
    Generate_randomInt [label="int randomInt = rand.nextInt(100); // Generate random integer between 0 and 99"];
    Generate_randomDouble [label="double randomDouble = rand.nextDouble(); // Generate random double between
0.0 and 1.0"];
    Print_randomInt [label="System.out.println(\"Random integer: \" + randomInt);"];
    Print_randomDouble [label="System.out.println(\"Random double: \" + randomDouble);"];
    End [label="End"];

    Start -> Declare_rand;
    Declare_rand -> Generate_randomInt;
    Generate_randomInt -> Generate_randomDouble;
    Generate_randomDouble -> Print_randomInt;
    Print_randomInt -> Print_randomDouble;
```

```
    Print_randomDouble -> End;
}
```

26. Print Fibonacci Series Using Recursion:

```
public class FibonacciRecursion {
    public static void main(String[] args) {
        int n = 10;
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }
    static int fibonacci(int n) {
        if (n <= 1)
            return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Properly formatted dot string for the Above Print Fibonacci Series Using Recursion is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_n [label="int n = 10;"];
    Loop_condition [label="for (int i = 0; i < n; i++)"];
    Call_fibonacci [label="fibonacci(i)"];
    Check_base_case [label="if (n <= 1)\n    return n;"];
    Recursive_call_1 [label="fibonacci(n - 1)"];
    Recursive_call_2 [label="fibonacci(n - 2)"];
    Return_value [label="return fibonacci(n - 1) + fibonacci(n - 2);"];
    Print_fibonacci [label="System.out.print(fibonacci(i) + \" \");"];
    End [label="End"];

    Start -> Declare_n;
    Declare_n -> Loop_condition;
    Loop_condition -> Call_fibonacci [label="true"];
    Call_fibonacci -> Check_base_case;
    Check_base_case -> Return_value [label="n <= 1"];
    Check_base_case -> Recursive_call_1 [label="n > 1"];
    Recursive_call_1 -> Return_value;
    Recursive_call_2 -> Return_value;
    Return_value -> Print_fibonacci;
    Print_fibonacci -> Loop_condition;
    Loop_condition -> End [label="i >= n"];
}
```

27. Find GCD (Greatest Common Divisor):

```
public class GCD {
    public static void main(String[] args) {
        int num1 = 12, num2 = 18;
        while (num1 != num2) {
            if (num1 > num2)
```

```
            num1 -= num2;
        else
            num2 -= num1;
    }
    System.out.println("GCD: " + num1);
  }
}
```

Properly formatted dot string for the Above Find GCD is:

```
digraph ControlFlowDiagram {
  node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
  Start [label="Start"];
  Declare_num1 [label="int num1 = 12;"];
  Declare_num2 [label="int num2 = 18;"];
  Loop_condition [label="while (num1 != num2)"];
  Check_num1_greater [label="if (num1 > num2)\n    num1 -= num2;"];
  Check_num2_greater [label="else\n    num2 -= num1;"];
  Print_GCD [label="System.out.println(\"GCD: \" + num1);"];
  End [label="End"];

  Start -> Declare_num1;
  Declare_num1 -> Declare_num2;
  Declare_num2 -> Loop_condition;
  Loop_condition -> Check_num1_greater [label="true"];
  Check_num1_greater -> Loop_condition;
  Loop_condition -> Check_num2_greater [label="false"];
  Check_num2_greater -> Loop_condition;
  Loop_condition -> Print_GCD [label="num1 == num2"];
  Print_GCD -> End;
}
```

28. Count Number of Words in a Sentence:

```
public class WordCount {
  public static void main(String[] args) {
    String sentence = "Java programming is fun";
    int count = 1;
    for (int i = 0; i < sentence.length(); i++) {
      if (sentence.charAt(i) == ' ')
        count++;
    }
    System.out.println("Number of words: " + count);
  }
}
```

Properly formatted dot string for the Above Count Number of Words in a Sentence is:

```
digraph ControlFlowDiagram {
  node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
  Start [label="Start"];
  Declare_sentence [label="String sentence = \"Java programming is fun\";"];
```

```
    Initialize_count [label="int count = 1;"];
    Loop_condition [label="for (int i = 0; i < sentence.length(); i++)"];
    Check_space [label="if (sentence.charAt(i) == ' ')\n    count++;"];
    Print_count [label="System.out.println(\"Number of words: \" + count);"];
    End [label="End"];

    Start -> Declare_sentence;
    Declare_sentence -> Initialize_count;
    Initialize_count -> Loop_condition;
    Loop_condition -> Check_space [label="true"];
    Check_space -> Loop_condition;
    Loop_condition -> Print_count [label="i == sentence.length()"];
    Print_count -> End;
}
```

29. Check Palindrome:

```
public class Palindrome {
    public static void main(String[] args) {
        String str = "radar";
        boolean isPalindrome = true;
        for (int i = 0; i < str.length() / 2; i++) {
            if (str.charAt(i) != str.charAt(str.length() - i - 1)) {
                isPalindrome = false;
                break;
            }
        }
        if (isPalindrome)
            System.out.println(str + " is a palindrome.");
        else
            System.out.println(str + " is not a palindrome.");
    }
}
```

Properly formatted dot string for the Above Check Palindrome is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_str [label="String str = \"radar\";"];
    Initialize_isPalindrome [label="boolean isPalindrome = true;"];
    Loop_condition [label="for (int i = 0; i < str.length() / 2; i++)"];
    Check_characters [label="if (str.charAt(i) != str.charAt(str.length() - i - 1)) {\n    isPalindrome = false;\n    break;\n}"];
    If_block [label="if (isPalindrome)"];
    If_action [label="System.out.println(str + \" is a palindrome.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(str + \" is not a palindrome.\");"];
    End [label="End"];

    Start -> Declare_str;
    Declare_str -> Initialize_isPalindrome;
    Initialize_isPalindrome -> Loop_condition;
    Loop_condition -> Check_characters [label="true"];
```

```
    Check_characters -> Loop_condition;
    Loop_condition -> If_block [label="i == str.length() / 2"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

30. Convert Decimal to Binary:

```
public class DecimalToBinary {
    public static void main(String[] args) {
        int decimal = 10;
        StringBuilder binary = new StringBuilder();
        while (decimal > 0) {
            binary.insert(0, decimal % 2);
            decimal /= 2;
        }
        System.out.println("Binary representation: " + binary);
    }
}
```

Properly formatted dot string for the Above Convert Decimal to Binary is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_decimal [label="int decimal = 10;"];
    Initialize_binary [label="StringBuilder binary = new StringBuilder();"];
    Loop_condition [label="while (decimal > 0)"];
    Calculate_binary_digit [label="binary.insert(0, decimal % 2);\ndecimal /= 2;"];
    Print_binary [label="System.out.println(\"Binary representation: \" + binary);"];
    End [label="End"];

    Start -> Declare_decimal;
    Declare_decimal -> Initialize_binary;
    Initialize_binary -> Loop_condition;
    Loop_condition -> Calculate_binary_digit [label="true"];
    Calculate_binary_digit -> Loop_condition;
    Loop_condition -> Print_binary [label="decimal <= 0"];
    Print_binary -> End;
}
```

31. Convert Binary to Decimal:

```
public class BinaryToDecimal {
    public static void main(String[] args) {
        String binary = "1010";
        int decimal = Integer.parseInt(binary, 2);
        System.out.println("Decimal representation: " + decimal);
    }
```

```
}
```

Properly formatted dot string for the Above Convert Binary to Decimal is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_binary [label="String binary = \"1010\";"];
    Convert_to_decimal [label="int decimal = Integer.parseInt(binary, 2);"];
    Print_decimal [label="System.out.println(\"Decimal representation: \" + decimal);"];
    End [label="End"];

    Start -> Declare_binary;
    Declare_binary -> Convert_to_decimal;
    Convert_to_decimal -> Print_decimal;
    Print_decimal -> End;
}
```

32. Check Armstrong Number:

```
public class ArmstrongNumber {
    public static void main(String[] args) {
        int num = 153;
        int originalNum = num;
        int result = 0;
        while (num != 0) {
            int remainder = num % 10;
            result += Math.pow(remainder, 3);
            num /= 10;
        }
        if (result == originalNum)
            System.out.println(originalNum + " is an Armstrong number.");
        else
            System.out.println(originalNum + " is not an Armstrong number.");
    }
}
```

Properly formatted dot string for the Above Check Armstrong Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num [label="int num = 153;"];
    Assign_originalNum [label="int originalNum = num;"];
    Initialize_result [label="int result = 0;"];
    Loop_condition [label="while (num != 0)"];
    Calculate_remainder [label="int remainder = num % 10;"];
    Calculate_result [label="result += Math.pow(remainder, 3);"];
    Update_num [label="num /= 10;"];
    If_block [label="if (result == originalNum)"];
    If_action [label="System.out.println(originalNum + \" is an Armstrong number.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(originalNum + \" is not an Armstrong number.\");"];
```

```
    End [label="End"];

    Start -> Declare_num;
    Declare_num -> Assign_originalNum;
    Assign_originalNum -> Initialize_result;
    Initialize_result -> Loop_condition;
    Loop_condition -> Calculate_remainder [label="true"];
    Calculate_remainder -> Calculate_result;
    Calculate_result -> Update_num;
    Update_num -> Loop_condition;
    Loop_condition -> If_block [label="num == 0"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

33. Find Factorial Using Recursion:

```
public class FactorialRecursion {
    public static void main(String[] args) {
        int num = 5;
        System.out.println("Factorial of " + num + " = " + factorial(num));
    }
    static int factorial(int n) {
        if (n == 0)
            return 1;
        else
            return n * factorial(n - 1);
    }
}
```

Properly formatted dot string for the Above Find Factorial Using Recursion is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num [label="int num = 5;"];
    Call_factorial [label="factorial(num)"];
    Check_base_case [label="if (n == 0)\n    return 1;"];
    Recursive_call [label="return n * factorial(n - 1);"];
    Print_factorial [label="System.out.println(\"Factorial of \" + num + \" = \" + factorial(num));"];
    End [label="End"];

    Start -> Declare_num;
    Declare_num -> Call_factorial;
    Call_factorial -> Check_base_case;
    Check_base_case -> Recursive_call [label="n != 0"];
    Recursive_call -> Check_base_case;
    Check_base_case -> Print_factorial;
```

```
    Print_factorial -> End;
}
```

34. Check Perfect Number:

```
public class PerfectNumber {
    public static void main(String[] args) {
        int num = 28;
        int sum = 0;
        for (int i = 1; i <= num / 2; i++) {
            if (num % i == 0)
                sum += i;
        }
        if (sum == num)
            System.out.println(num + " is a perfect number.");
        else
            System.out.println(num + " is not a perfect number.");
```

Properly formatted dot string for the Above Check Perfect Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num [label="int num = 28;"];
    Initialize_sum [label="int sum = 0;"];
    Loop_condition [label="for (int i = 1; i <= num / 2; i++)"];
    Check_divisibility [label="if (num % i == 0)\n    sum += i;"];
    If_block [label="if (sum == num)"];
    If_action [label="System.out.println(num + \" is a perfect number.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(num + \" is not a perfect number.\");"];
    End [label="End"];

    Start -> Declare_num;
    Declare_num -> Initialize_sum;
    Initialize_sum -> Loop_condition;
    Loop_condition -> Check_divisibility [label="true"];
    Check_divisibility -> Loop_condition;
    Loop_condition -> If_block [label="i > num / 2"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

35. Implementing a Binary Search Tree (BST)

```
class TreeNode {
    int val;
```

```
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    private TreeNode root;

    public BinarySearchTree() {
        this.root = null;
    }

    public void insert(int val) {
        this.root = insertNode(this.root, val);
    }

    private TreeNode insertNode(TreeNode root, int val) {
        if (root == null) {
            return new TreeNode(val);
        }
        if (val < root.val) {
            root.left = insertNode(root.left, val);
        } else if (val > root.val) {
            root.right = insertNode(root.right, val);
        }
        return root;
    }

    public boolean search(int val) {
        return searchNode(this.root, val);
    }

    private boolean searchNode(TreeNode root, int val) {
        if (root == null) {
            return false;
        }
        if (root.val == val) {
            return true;
        }
        if (val < root.val) {
            return searchNode(root.left, val);
        } else {
            return searchNode(root.right, val);
        }
    }

    public void inorderTraversal() {
        performInorderTraversal(this.root);
    }
```

```
    private void performInorderTraversal(TreeNode root) {
        if (root != null) {
            performInorderTraversal(root.left);
            System.out.print(root.val + " ");
            performInorderTraversal(root.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        bst.insert(50);
        bst.insert(30);
        bst.insert(70);
        bst.insert(20);
        bst.insert(40);
        bst.insert(60);
        bst.insert(80);

        System.out.println("Inorder Traversal:");
        bst.inorderTraversal();

        int searchKey = 40;
        System.out.println("\nIs " + searchKey + " present in BST? " + bst.search(searchKey));
    }
}
```

Properly formatted dot string for the Above Implementing a Binary Search Tree is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_bst [label="BinarySearchTree bst = new BinarySearchTree();"];
    Insert_50 [label="bst.insert(50);"];
    Insert_30 [label="bst.insert(30);"];
    Insert_70 [label="bst.insert(70);"];
    Insert_20 [label="bst.insert(20);"];
    Insert_40 [label="bst.insert(40);"];
    Insert_60 [label="bst.insert(60);"];
    Insert_80 [label="bst.insert(80);"];
    Inorder_traversal [label="System.out.println(\"Inorder Traversal:\");\nbst.inorderTraversal();"];
    Declare_searchKey [label="int searchKey = 40;"];
    Search_40 [label="System.out.println(\"\\nIs \" + searchKey + \" present in BST? \" + bst.search(searchKey));"];
    End [label="End"];

    Start -> Declare_bst;
    Declare_bst -> Insert_50;
    Insert_50 -> Insert_30;
    Insert_30 -> Insert_70;
    Insert_70 -> Insert_20;
    Insert_20 -> Insert_40;
    Insert_40 -> Insert_60;
    Insert_60 -> Insert_80;
    Insert_80 -> Inorder_traversal;
```

```
    Inorder_traversal -> Declare_searchKey;
    Declare_searchKey -> Search_40;
    Search_40 -> End;
}
```

36. Implementing a Linked List with various operations:

```
class ListNode {
    int val;
    ListNode next;

    public ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class LinkedList {
    private ListNode head;

    public LinkedList() {
        this.head = null;
    }

    public void insertAtBeginning(int val) {
        ListNode newNode = new ListNode(val);
        newNode.next = this.head;
        this.head = newNode;
    }

    public void insertAtEnd(int val) {
        ListNode newNode = new ListNode(val);
        if (this.head == null) {
            this.head = newNode;
            return;
        }
        ListNode current = this.head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }

    public void delete(int val) {
        if (this.head == null) {
            return;
        }
        if (this.head.val == val) {
            this.head = this.head.next;
            return;
        }
        ListNode current = this.head;
        while (current.next != null && current.next.val != val) {
```

```
            current = current.next;
        }
        if (current.next != null) {
            current.next = current.next.next;
        }
    }

    public void display() {
        ListNode current = this.head;
        while (current != null) {
            System.out.print(current.val + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.insertAtEnd(10);
        list.insertAtEnd(20);
        list.insertAtBeginning(5);
        list.insertAtEnd(30);
        list.display();
        list.delete(20);
        list.display();
    }
}
```

Properly formatted dot string for the Above Implementing a Linked List with various operations is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_list [label="LinkedList list = new LinkedList();"];
    Insert_end [label="list.insertAtEnd(10);\nlist.insertAtEnd(20);\nlist.insertAtEnd(30);"];
    Insert_beginning [label="list.insertAtBeginning(5);"];
    Display_initial [label="list.display();"];
    Delete_element [label="list.delete(20);"];
    Display_final [label="list.display();"];
    End [label="End"];

    Start -> Declare_list;
    Declare_list -> Insert_end;
    Insert_end -> Insert_beginning;
    Insert_beginning -> Display_initial;
    Display_initial -> Delete_element;
    Delete_element -> Display_final;
    Display_final -> End;
}
```

37. Implementing a simple calculator using Object-Oriented Programming (OOP) principles:

```
import java.util.Scanner;

interface Operation {
    double perform(double operand1, double operand2);
}

class Add implements Operation {
    public double perform(double operand1, double operand2) {
        return operand1 + operand2;
    }
}

class Subtract implements Operation {
    public double perform(double operand1, double operand2) {
        return operand1 - operand2;
    }
}

class Multiply implements Operation {
    public double perform(double operand1, double operand2) {
        return operand1 * operand2;
    }
}

class Divide implements Operation {
    public double perform(double operand1, double operand2) {
        if (operand2 == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return operand1 / operand2;
    }
}

public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = scanner.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();
        System.out.print("Enter operator (+, -, *, /): ");
        char operator = scanner.next().charAt(0);
        double result;
        Operation operation;
        switch (operator) {
            case '+':
                operation = new Add();
                break;
            case '-':
                operation = new Subtract();
                break;
            case '*':
```

```
                operation = new Multiply();
                break;
            case '/':
                operation = new Divide();
                break;
            default:
                System.out.println("Invalid operator");
                return;
        }
        result = operation.perform(num1, num2);
        System.out.println("Result: " + result);
    }
}
```

Properly formatted dot string for the Above Implementing a simple calculator using Object-Oriented Programming (OOP) principles is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_scanner [label="Scanner scanner = new Scanner(System.in);"];
    Input_num1 [label="Enter first number:"];
    Read_num1 [label="double num1 = scanner.nextDouble();"];
    Input_num2 [label="Enter second number:"];
    Read_num2 [label="double num2 = scanner.nextDouble();"];
    Input_operator [label="Enter operator (+, -, *, /):"];
    Read_operator [label="char operator = scanner.next().charAt(0);"];
    Declare_result [label="double result;"];
    Declare_operation [label="Operation operation;"];
    Switch_operator [label="switch (operator)"];
    Case_add [label="case '+':\noperation = new Add();"];
    Case_subtract [label="case '-':\noperation = new Subtract();"];
    Case_multiply [label="case '*':\noperation = new Multiply();"];
    Case_divide [label="case '/':\noperation = new Divide();"];
    Default_case [label="default:\nSystem.out.println(\"Invalid operator\");\nreturn;"];
    Perform_operation [label="result = operation.perform(num1, num2);"];
    Print_result [label="System.out.println(\"Result: \" + result);"];
    End [label="End"];

    Start -> Declare_scanner;
    Declare_scanner -> Input_num1;
    Input_num1 -> Read_num1;
    Read_num1 -> Input_num2;
    Input_num2 -> Read_num2;
    Read_num2 -> Input_operator;
    Input_operator -> Read_operator;
    Read_operator -> Declare_result;
    Declare_result -> Declare_operation;
    Declare_operation -> Switch_operator;
    Switch_operator -> Case_add [label="+"];
    Switch_operator -> Case_subtract [label="-"];
    Switch_operator -> Case_multiply [label="*"];
    Switch_operator -> Case_divide [label="/"];
    Switch_operator -> Default_case [label="default"];
```

```
    Case_add -> Perform_operation -> Print_result;
    Case_subtract -> Perform_operation -> Print_result;
    Case_multiply -> Perform_operation -> Print_result;
    Case_divide -> Perform_operation -> Print_result;
    Default_case -> End;
}
```

38. Matrix Multiplication:

```
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] matrix1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int[][] matrix2 = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
        int[][] result = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        System.out.println("Resultant Matrix after Multiplication:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Properly formatted dot string for the Above Matrix Multiplication is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_matrices [label="Declare matrices and initialize"];
    Initialize_result [label="Initialize result matrix"];
    Loop_i [label="for (int i = 0; i < 3; i++)"];
    Loop_j [label="for (int j = 0; j < 3; j++)"];
    Loop_k [label="for (int k = 0; k < 3; k++)"];
    Multiply [label="result[i][j] += matrix1[i][k] * matrix2[k][j];"];
    Print_result [label="Print resultant matrix"];
    End [label="End"];

    Start -> Declare_matrices;
    Declare_matrices -> Initialize_result;
    Initialize_result -> Loop_i;
    Loop_i -> Loop_j [label="true"];
    Loop_j -> Loop_k [label="true"];
    Loop_k -> Multiply [label="true"];
    Multiply -> Loop_k [label="true"];
    Loop_k -> Loop_j [label="false"];
```

```
    Loop_j -> Loop_i [label="false"];
    Loop_i -> Print_result [label="false"];
    Print_result -> End;
}
```

39. Fibonacci Series with Dynamic Programming:

```
public class FibonacciDynamic {
    public static void main(String[] args) {
        int n = 10;
        int[] fibonacci = new int[n];
        fibonacci[0] = 0;
        fibonacci[1] = 1;

        for (int i = 2; i < n; i++) {
            fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
        }

        System.out.println("Fibonacci Series:");
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacci[i] + " ");
        }
    }
}
```

Properly formatted dot string for the Above Fibonacci Series with Dynamic Programming is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_n [label="Initialize n"];
    Declare_array [label="Declare and initialize Fibonacci array"];
    Loop_i [label="for (int i = 2; i < n; i++)"];
    Calculate_fibonacci [label="fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];"];
    Print_fibonacci [label="Print Fibonacci series"];
    End [label="End"];

    Start -> Initialize_n;
    Initialize_n -> Declare_array;
    Declare_array -> Loop_i;
    Loop_i -> Calculate_fibonacci [label="true"];
    Calculate_fibonacci -> Loop_i;
    Loop_i -> Print_fibonacci [label="false"];
    Print_fibonacci -> End;
}
```

40. Printing Pascal's Triangle:

```
public class PascalTriangle {
    public static void main(String[] args) {
        int rows = 5;
        for (int i = 0; i < rows; i++) {
            int number = 1;
            for (int j = 0; j <= i; j++) {
```

```
            System.out.print(number + " ");
            number = number * (i - j) / (j + 1);
        }
        System.out.println();
    }
  }
}
```

Properly formatted dot string for the Above Printing Pascal's Triangle is:

```
digraph ControlFlowDiagram {
  node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
  Start [label="Start"];
  Initialize_rows [label="Initialize rows"];
  Loop_i [label="for (int i = 0; i < rows; i++)"];
  Initialize_number [label="Initialize number to 1"];
  Loop_j [label="for (int j = 0; j <= i; j++)"];
  Print_number [label="Print number"];
  Calculate_next_number [label="Calculate next number"];
  Update_number [label="Update number"];

  Start -> Initialize_rows;
  Initialize_rows -> Loop_i;
  Loop_i -> Initialize_number [label="true"];
  Initialize_number -> Loop_j;
  Loop_j -> Print_number [label="true"];
  Print_number -> Calculate_next_number;
  Calculate_next_number -> Update_number;
  Update_number -> Loop_j [label="true"];
  Loop_j -> Loop_i [label="false"];
  Loop_i -> End [label="false"];
  End [label="End"];
}
```

41. Printing a Diamond Pattern:

```
public class DiamondPattern {
  public static void main(String[] args) {
    int n = 5;
    int spaces = n - 1;
    int stars = 1;

    for (int i = 1; i <= n * 2 - 1; i++) {
      for (int j = 1; j <= spaces; j++) {
        System.out.print(" ");
      }
      for (int j = 1; j <= stars; j++) {
        System.out.print("*");
      }
      System.out.println();

      if (i < n) {
        spaces--;
        stars += 2;
      } else {
```

```
            spaces++;
            stars -= 2;
        }
    }
}
```

Properly formatted dot string for the Above Printing a Diamond Pattern is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables: n, spaces, stars"];
    Loop_i [label="for (int i = 1; i <= n * 2 - 1; i++)"];
    Loop_spaces [label="for (int j = 1; j <= spaces; j++)"];
    Print_space [label="Print space"];
    Loop_stars [label="for (int j = 1; j <= stars; j++)"];
    Print_star [label="Print star"];
    Update_spaces_stars [label="Update spaces and stars"];
    Check_condition [label="if (i < n)"];
    Update_spaces_stars_else [label="Update spaces and stars"];

    Start -> Initialize_variables;
    Initialize_variables -> Loop_i;
    Loop_i -> Loop_spaces [label="true"];
    Loop_spaces -> Print_space;
    Print_space -> Loop_spaces [label="true"];
    Loop_spaces -> Loop_stars [label="false"];
    Loop_stars -> Print_star;
    Print_star -> Loop_stars [label="true"];
    Loop_stars -> Update_spaces_stars [label="false"];
    Update_spaces_stars -> Check_condition;
    Check_condition -> Update_spaces_stars_else [label="true"];
    Update_spaces_stars_else -> Loop_i;
    Loop_i -> End [label="false"];
    End [label="End"];
}
```

42. Printing Prime Numbers up to a Given Number:

```
public class PrimeNumbers {
    public static void main(String[] args) {
        int n = 50;

        for (int i = 2; i <= n; i++) {
            boolean isPrime = true;
            for (int j = 2; j <= Math.sqrt(i); j++) {
                if (i % j == 0) {
                    isPrime = false;
                    break;
                }
            }
```

```
            if (isPrime) {
                System.out.print(i + " ");
            }
        }
    }
}
```

Properly formatted dot string for the Above Printing Prime Numbers up to a Given Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_n [label="Initialize n"];
    Loop_i [label="for (int i = 2; i <= n; i++)"];
    Initialize_isPrime [label="Initialize isPrime"];
    Loop_j [label="for (int j = 2; j <= Math.sqrt(i); j++)"];
    Check_prime [label="if (i % j == 0)"];
    Update_isPrime [label="Update isPrime"];
    Print_prime [label="Print prime number"];

    Start -> Initialize_n;
    Initialize_n -> Loop_i;
    Loop_i -> Initialize_isPrime;
    Initialize_isPrime -> Loop_j;
    Loop_j -> Check_prime [label="true"];
    Check_prime -> Update_isPrime [label="true"];
    Update_isPrime -> Loop_j;
    Check_prime -> Loop_j [label="false"];
    Loop_j -> Loop_i [label="false"];
    Loop_i -> Print_prime [label="true"];
    Print_prime -> Loop_i;
    Loop_i -> End [label="false"];
    End [label="End"];
}
```

43. Printing the Factorial of a Number Using a Loop:

```
public class FactorialLoop {
    public static void main(String[] args) {
        int n = 5;
        int factorial = 1;

        for (int i = 1; i <= n; i++) {
            factorial *= i;
        }

        System.out.println("Factorial of " + n + " = " + factorial);
    }
}
```

Properly formatted dot string for the Above Printing the Factorial of a Number Using a Loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
```

```
    Start [label="Start"];
    Initialize_n [label="Initialize n"];
    Initialize_factorial [label="Initialize factorial"];
    Loop [label="for (int i = 1; i <= n; i++)"];
    Update_factorial [label="Update factorial"];
    Print_result [label="Print factorial"];
    End [label="End"];

    Start -> Initialize_n;
    Initialize_n -> Initialize_factorial;
    Initialize_factorial -> Loop;
    Loop -> Update_factorial [label="true"];
    Update_factorial -> Loop;
    Loop -> Print_result [label="false"];
    Print_result -> End;
}
```

44. Counting the Number of Words in a Sentence Using a Loop:

```
public class WordCountLoop {
    public static void main(String[] args) {
        String sentence = "Java programming is interesting";
        int count = 1;

        for (int i = 0; i < sentence.length(); i++) {
            if (sentence.charAt(i) == ' ') {
                count++;
            }
        }

        System.out.println("Number of words: " + count);
    }
}
```

Properly formatted dot string for the Above Counting the Number of Words in a Sentence Using a Loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_sentence [label="Initialize sentence"];
    Initialize_count [label="Initialize count"];
    Loop [label="for (int i = 0; i < sentence.length(); i++)"];
    Check_character [label="if (sentence.charAt(i) == ' ')"];
    Increment_count [label="Increment count"];
    End [label="End"];

    Start -> Initialize_sentence;
    Initialize_sentence -> Initialize_count;
    Initialize_count -> Loop;
    Loop -> Check_character [label="true"];
    Check_character -> Increment_count [label="true"];
    Increment_count -> Loop;
    Loop -> End [label="false"];
```

```
}
```

45. Finding the GCD of Two Numbers Using Euclid's Algorithm:

```
public class GCDAlgorithm {
    public static void main(String[] args) {
        int num1 = 48, num2 = 18;

        while (num2 != 0) {
            int temp = num2;
            num2 = num1 % num2;
            num1 = temp;
        }

        System.out.println("GCD: " + num1);
    }
}
```

Properly formatted dot string for the Above Finding the GCD of Two Numbers Using Euclid's Algorithm is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables"];
    Loop [label="while (num2 != 0)"];
    Swap_variables [label="Swap variables"];
    Update_num2 [label="Update num2"];
    Update_num1 [label="Update num1"];
    End [label="End"];

    Start -> Initialize_variables;
    Initialize_variables -> Loop;
    Loop -> Swap_variables [label="true"];
    Swap_variables -> Update_num2;
    Update_num2 -> Loop;
    Loop -> Update_num1 [label="false"];
    Update_num1 -> End;
}
```

46. Calculating the Sum of Natural Numbers Using a Loop:

```
public class SumOfNaturalNumbers {
    public static void main(String[] args) {
        int n = 10;
        int sum = 0;

        for (int i = 1; i <= n; i++) {
            sum += i;
        }

        System.out.println("Sum of first " + n + " natural numbers: " + sum);
    }
}
```

Properly formatted dot string for the Above Calculating the Sum of Natural Numbers Using a Loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables"];
    Loop [label="for (int i = 1; i <= n; i++)"];
    Update_sum [label="Update sum"];
    End [label="End"];

    Start -> Initialize_variables;
    Initialize_variables -> Loop;
    Loop -> Update_sum [label="true"];
    Update_sum -> Loop;
    Loop -> End [label="false"];
}
```

47. Nested loop:

```
class Main {
  public static void main(String[] args) {

    int weeks = 3;
    int days = 7;

    // outer loop prints weeks
    for (int i = 1; i <= weeks; ++i) {
      System.out.println("Week: " + i);

      // inner loop prints days
      for (int j = 1; j <= days; ++j) {
        System.out.println("  Day: " + j);
      }
    }
  }
}
```

Properly formatted dot string for the Above Nested loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables"];
    Outer_Loop [label="for (int i = 1; i <= weeks; ++i)"];
    Print_Week [label="Print week"];
    Inner_Loop [label="for (int j = 1; j <= days; ++j)"];
    Print_Day [label="Print day"];
    End [label="End"];

    Start -> Initialize_variables;
    Initialize_variables -> Outer_Loop;
    Outer_Loop -> Print_Week [label="true"];
```

```
    Print_Week -> Inner_Loop;
    Inner_Loop -> Print_Day [label="true"];
    Print_Day -> Inner_Loop;
    Inner_Loop -> Outer_Loop [label="false"];
    Outer_Loop -> End [label="false"];
}
```

Training set for C# codes

The dot string response for the below number comparison C# code is
Number Comparison

```
int a = 10;
int b = 5;
int result;
if (a > b)
{
    result = a - b;
}
else
{
    result = b - a;
}
Console.WriteLine("Result: " + result);
```

Properly formatted dot string for the Above Number Comparison code

```
digraph {
    start -> assign_a
    assign_a -> assign_b
    assign_b -> condition
    condition -> code_block1 [label="true"]
    condition -> else_block [label="false"]
    code_block1 -> assign_result
    assign_result -> print
    else_block -> assign_result2
    assign_result2 -> print
    print -> end
}
```

The dot string response for the below Simple loop C# code is
Simple Loop

```
int n = 5;
for (int i = 0; i < n; i++)
{
    Console.WriteLine(i);
}
```

Properly formatted dot string for the Above Simple loop code is

```
digraph {
    start -> assign_n
    assign_n -> assign_i
    assign_i -> condition
    condition -> code_block [label="true"]
    condition -> end [label="false"]
    code_block -> increment_i
    increment_i -> condition
}
```

The dot string response for the below While loop code is
While loop

```
int n = 5;
int i = 0;
while (i < n)
{
    Console.WriteLine(i);
    i++;
}
```

Properly formatted dot string for the Above while loop code is

```
digraph {
    start -> assign_n
    assign_n -> assign_i
    assign_i -> condition
    condition -> code_block [label="true"]
    condition -> end [label="false"]
    code_block -> increment_i
    increment_i -> condition
}
```

The dot string response for the below Nested loop code is

Nested loop

```
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        Console.WriteLine(i + " " + j);
    }
}
```

Properly formatted dot string for the Above Nested loop code is

```
digraph {
    start -> assign_i1
    assign_i1 -> condition1
    condition1 -> code_block1 [label="true"]
    condition1 -> end [label="false"]
    code_block1 -> assign_j1
    assign_j1 -> condition2
    condition2 -> code_block2 [label="true"]
    condition2 -> increment_j [label="false"]
    increment_j -> condition2
    code_block2 -> increment_j
    increment_j -> condition2
    code_block2 -> increment_i
    increment_i -> condition1
}
```

The dot string response for the below Switch cases code is

Switch cases

```
int day = 3;
string dayName;
switch (day)
{
    case 1:
        dayName = "Monday";
        break;
    case 2:
        dayName = "Tuesday";
        break;
    case 3:
        dayName = "Wednesday";
        break;
```

```
    default:
        dayName = "Unknown";
        break;
}
Console.WriteLine("Day is: " + dayName);
```

Properly formatted dot string for the Above switch cases code is

```
digraph {
    start -> assign_day
    assign_day -> switch_start
    switch_start -> case1 [label="1"]
    switch_start -> case2 [label="2"]
    switch_start -> case3 [label="3"]
    switch_start -> default [label="default"]
    case1 -> assign_dayName1
    case2 -> assign_dayName2
    case3 -> assign_dayName3
    default -> assign_dayName_default
    assign_dayName1 -> print
    assign_dayName2 -> print
    assign_dayName3 -> print
    assign_dayName_default -> print
    print -> end
}
```

The dot string response for the below Try Catch finally code is

Try Catch finally

```
int num = 0;
try
{
    int result = 10 / num;
    Console.WriteLine("Result: " + result);
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Error: " + ex.Message);
}
finally
{
    Console.WriteLine("Finally block executed.");
}
```

Properly formatted dot string for the Try Catch finally code is

```
digraph {
    start -> try_start
    try_start -> divide [label="try"]
    divide -> print_result [label="success"]
    divide -> catch_start [label="failure"]
    print_result -> finally_start
    catch_start -> print_error
    print_error -> finally_start
    finally_start -> end
}
```

The dot string response for the below Method Invocation code is

Method Invocation code

```
int x = 5;
int y = 10;
int sum = Add(x, y);
Console.WriteLine("Sum: " + sum);
int Add(int a, int b)
```

```
{
    return a + b;
}
```
Properly formatted dot string for the Method Invocation code is
```
digraph {
    start -> assign_x
    assign_x -> assign_y
    assign_y -> invoke_Add
    invoke_Add -> declare_Add
    declare_Add -> return_Add
    return_Add -> print
    print -> end
}
```
The dot string response for the below Recursive method code is
Recursive method code
```
int Factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * Factorial(n - 1);
}
int result = Factorial(5);
Console.WriteLine("Factorial: " + result);
```
Properly formatted dot string for the Recursive method code is
```
digraph {
    start -> invoke_Factorial
    invoke_Factorial -> condition [label="n != 0"]
    condition -> return_factorial [label="true"]
    condition -> return_1 [label="false"]
    return_factorial -> invoke_Factorial
    invoke_Factorial -> end [label="n == 0"]
    return_1 -> end
}
```
The dot string response for the below Multiple if code is
Multiple if code
```
int score = 75;
string grade;
if (score >= 90)
{
    grade = "A";
}
else if (score >= 80)
{
    grade = "B";
}
else if (score >= 70)
{
    grade = "C";
}
else
{
    grade = "D";
}
Console.WriteLine("Grade: " + grade);
```
Properly formatted dot string for the Multiple if code
```
digraph {
```

```
    start -> condition1
    condition1 -> code_block1 [label="score >= 90"]
    condition1 -> condition2 [label="score < 90"]
    code_block1 -> assign_grade_A
    assign_grade_A -> print
    condition2 -> condition3 [label="score >= 80"]
    condition2 -> condition4 [label="score < 80"]
    condition3 -> code_block2 [label="score >= 80"]
    condition3 -> condition4 [label="score < 80"]
    code_block2 -> assign_grade_B
    assign_grade_B -> print
    condition4 -> condition5 [label="score >= 70"]
    condition4 -> assign_grade_D [label="score < 70"]
    condition5 -> code_block3 [label="score >= 70"]
    condition5 -> assign_grade_D [label="score < 70"]
    code_block3 -> assign_grade_C
    assign_grade_C -> print
    assign_grade_D -> print
    print -> end
}
```

The dot string response for the below handling Array code is

Handling Array code

```
int[] numbers = { 1, 2, 3, 4, 5 };
int sum = 0;
foreach (int num in numbers)
{
    sum += num;
}
Console.WriteLine("Sum: " + sum);
```

Properly formatted dot string for the handling Array code is

```
digraph {
    start -> declare_array
    declare_array -> iterate_array
    iterate_array -> code_block
    code_block -> increment_sum
    increment_sum -> iterate_array
    iterate_array -> end [label="end of array"]
    end -> print
    print -> end
}
```

The dot string response for the below enums code is

Enums code

```
enum Days { Monday, Tuesday, Wednesday, Thursday, Friday };
Days myDay = Days.Monday;
switch (myDay)
{
    case Days.Monday:
        Console.WriteLine("It's Monday!");
        break;
    case Days.Tuesday:
        Console.WriteLine("It's Tuesday!");
        break;
    default:
        Console.WriteLine("It's not Monday or Tuesday.");
        break;
}
```

Properly formatted dot string for the enums code is

```
digraph {
    start -> assign_myDay
    assign_myDay -> switch_start
    switch_start -> case_Monday [label="Monday"]
    switch_start -> case_Tuesday [label="Tuesday"]
    switch_start -> default [label="default"]
    case_Monday -> print_Monday
    case_Tuesday -> print_Tuesday
    default -> print_default
    print_Monday -> end
    print_Tuesday -> end
    print_default -> end
}
```

The dot string response for the below working with classes code is

working with classes

```
class Car
{
    public string Make { get; set; }
    public string Model { get; set; }
    public void Start()
    {
        Console.WriteLine("The car is starting.");
    }
}
Car myCar = new Car();
myCar.Make = "Toyota";
myCar.Model = "Camry";
myCar.Start();
```

Properly formatted dot string for the working with classes code is

```
digraph {
    start -> create_object
    create_object -> assign_Make
    assign_Make -> assign_Model
    assign_Model -> invoke_Start
    invoke_Start -> print
    print -> end
}
```

The dot string response for the below String manipulation code is

String manipulation

```
string str = "Hello, world!";
int length = str.Length;
string upperCase = str.ToUpper();
string lowerCase = str.ToLower();
Console.WriteLine("Length: " + length);
Console.WriteLine("Upper case: " + upperCase);
Console.WriteLine("Lower case: " + lowerCase);
```

Properly formatted dot string for the String manipulation code is

```
digraph {
    start -> assign_str
    assign_str -> get_length
    get_length -> to_upper
    to_upper -> to_lower
    to_lower -> print_length
    print_length -> print_upper
    print_upper -> print_lower
    print_lower -> end
}
```

The dot string response for the below custom exceptions code is

```
custom exceptions
class MyCustomException : Exception
{
    public MyCustomException(string message) : base(message)
    {
    }
}
try
{
    throw new MyCustomException("This is a custom exception.");
}
catch (MyCustomException ex)
{
    Console.WriteLine("Custom Exception: " + ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine("Exception: " + ex.Message);
}
```

Properly formatted dot string for the custom exceptions code is

```
digraph {
    start -> try_start
    try_start -> throw_custom [label="try"]
    throw_custom -> catch_custom [label="MyCustomException"]
    catch_custom -> print_custom
    print_custom -> end
}
```

The dot string response for the below Interfaces code is

```
Interfaces
interface IShape
{
    double GetArea();
}
class Circle : IShape
{
    private double radius;
    public Circle(double radius)
    {
        this.radius = radius;
    }
    public double GetArea()
    {
        return Math.PI * radius * radius;
    }
}
Circle circle = new Circle(5);
Console.WriteLine("Circle Area: " + circle.GetArea());
```

Properly formatted dot string for the Interface code is

```
digraph {
    start -> create_object
    create_object -> invoke_GetArea
    invoke_GetArea -> print_area
    print_area -> end
}
```

The dot string response for the below Collections code is

Collections
```
List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
int sum = 0;
foreach (int num in numbers)
{
    sum += num;
}
Console.WriteLine("Sum: " + sum);
```
Properly formatted dot string for the Collections code is
```
digraph {
    start -> create_list
    create_list -> iterate_list
    iterate_list -> code_block
    code_block -> increment_sum
    increment_sum -> iterate_list
    iterate_list -> end [label="end of list"]
    end -> print
    print -> end
}
```
The dot string response for the below LINQ code is
LINQ
```
List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
var evenNumbers = numbers.Where(n => n % 2 == 0).ToList();
foreach (int num in evenNumbers)
{
    Console.WriteLine(num);
}
```
Properly formatted dot string for the LINQ code is
```
digraph {
    start -> create_list
    create_list -> linq_query
    linq_query -> print
    print -> end
}
```
The dot string response for the below handling files operations code is
handling files operations
```
string path = "example.txt";
string[] lines = { "Line 1", "Line 2", "Line 3" };
File.WriteAllLines(path, lines);
string[] readLines = File.ReadAllLines(path);
foreach (string line in readLines)
{
    Console.WriteLine(line);
}
```
Properly formatted dot string for the handling files operations code is
```
digraph {
    start -> write_lines
    write_lines -> read_lines
    read_lines -> print_lines
    print_lines -> end
}
```
The dot string response for the below Multi-threading code is
Multi-threading
```
using System;
using System.Threading.Tasks;
class Program
{
```

```
    static void Main()
    {
        Task task1 = Task.Run(() => DoWork(1));
        Task task2 = Task.Run(() => DoWork(2));
        Task.WaitAll(task1, task2);
        Console.WriteLine("All tasks completed.");
    }
    static void DoWork(int id)
    {
        Console.WriteLine("Task {0} is starting.", id);
        Task.Delay(1000).Wait();
        Console.WriteLine("Task {0} is done.", id);
    }
}
```

Properly formatted dot string for the Multi-threading code is

```
digraph {
    start -> task1_start
    start -> task2_start
    task1_start -> task1_work
    task2_start -> task2_work
    task1_work -> task1_done
    task2_work -> task2_done
    task1_done -> join
    task2_done -> join
    join -> print
    print -> end
}
```

The dot string response for the below Event driven code is

Event driven

```
using System;
class Program
{
    static void Main()
    {
        Button button = new Button();
        button.Click += (sender, e) => Console.WriteLine("Button clicked.");
        button.PerformClick();
    }
}
class Button
{
    public event EventHandler Click;
    public void PerformClick()
    {
        OnClick(EventArgs.Empty);
    }
    protected virtual void OnClick(EventArgs e)
    {
        Click?.Invoke(this, e);
    }
}
```

Properly formatted dot string for the Event driven code is

```
digraph {
    start -> create_button
    create_button -> add_event
    add_event -> invoke_click
    invoke_click -> print_message
```

```
    print_message -> end
}
```
The dot string response for the below Async wait time code is
Async wait time
```
using System;
using System.Net.Http;
using System.Threading.Tasks;
class Program
{
    static async Task Main()
    {
        await DoWorkAsync();
    }
    static async Task DoWorkAsync()
    {
        HttpClient client = new HttpClient();
        string result = await client.GetStringAsync("https://jsonplaceholder.typicode.com/todos/1");
        Console.WriteLine(result);
    }
}
```
Properly formatted dot string for the Async wait time code is
```
digraph {
    start -> create_client
    create_client -> get_data
    get_data -> print_result
    print_result -> end
}
```
The dot string response for the below Linear Search code is
Linear Search
```
using System;
class Program
{
    static void Main()
    {
        int[] array = { 10, 20, 30, 40, 50 };
        int target = 30;
        int index = LinearSearch(array, target);
        Console.WriteLine("Linear search result: " + index);
    }
    static int LinearSearch(int[] array, int target)
    {
        for (int i = 0; i < array.Length; i++)
        {
            if (array[i] == target)
            {
                return i;
            }
        }
        return -1; // Element not found
    }
}
```
Properly formatted dot string for the Linear Search code is
```
digraph {
    start -> loop_start
    loop_start -> condition
    condition -> found [label="array[i] == target"]
    condition -> iterate [label="array[i] != target"]
```

```
    found -> print_result
    iterate -> increment_i
    increment_i -> condition
    print_result -> end
}
```

The dot string response for the below Binary Search code is

```
using System;
class Program
{
    static void Main()
    {
        int[] array = { 10, 20, 30, 40, 50 };
        int target = 30;
        int index = BinarySearch(array, target);
        Console.WriteLine("Binary search result: " + index);
    }
    static int BinarySearch(int[] array, int target)
    {
        int left = 0;
        int right = array.Length - 1;

        while (left <= right)
        {
            int mid = left + (right - left) / 2;

            if (array[mid] == target)
            {
                return mid;
            }
            else if (array[mid] < target)
            {
                left = mid + 1;
            }
            else
            {
                right = mid - 1;
            }
        }

        return -1; // Element not found
    }
}
```

Binary Search

Properly formatted dot string for the Binary Search code is

```
digraph {
    start -> initialize_left_right
    initialize_left_right -> while_loop_start
    while_loop_start -> condition
    condition -> found [label="left <= right"]
    condition -> not_found [label="left > right"]
    found -> check_mid
    not_found -> return_not_found
    check_mid -> equal_target [label="array[mid] == target"]
    equal_target -> print_result
    equal_target -> adjust_right [label="array[mid] < target"]
    equal_target -> adjust_left [label="array[mid] > target"]
    adjust_right -> update_left
```

```
    adjust_left -> update_right
    update_left -> condition
    update_right -> condition
    print_result -> end
    return_not_found -> end
}
```

The dot string response for the below Bubble Sort code is

Bubble Sort

```
using System;
class Program
{
    static void Main()
    {
        int[] array = { 64, 25, 12, 22, 11 };
        BubbleSort(array);
        Console.WriteLine("Sorted array:");
        foreach (int num in array)
        {
            Console.Write(num + " ");
        }
    }
    static void BubbleSort(int[] array)
    {
        int n = array.Length;
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (array[j] > array[j + 1])
                {
                    // Swap array[j] and array[j+1]
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }
}
```

Properly formatted dot string for the Bubble Sort code is

```
digraph {
    start -> outer_loop_start
    outer_loop_start -> inner_loop_start
    inner_loop_start -> condition1
    condition1 -> condition2 [label="j < n - i - 1"]
    condition2 -> swap [label="array[j] > array[j + 1]"]
    swap -> increment_j
    increment_j -> condition1
    condition2 -> increment_i [label="j >= n - i - 1"]
    increment_i -> outer_loop_start
    condition1 -> end [label="j >= n - i - 1"]
    end -> print_sorted_array
    print_sorted_array -> end
}
```

The dot string response for the below Selection Sort code is

Selection Sort

```
using System;
```

```
class Program
{
    static void Main()
    {
        int[] array = { 64, 25, 12, 22, 11 };
        SelectionSort(array);
        Console.WriteLine("Sorted array:");
        foreach (int num in array)
        {
            Console.Write(num + " ");
        }
    }
    static void SelectionSort(int[] array)
    {
        int n = array.Length;
        for (int i = 0; i < n - 1; i++)
        {
            int minIndex = i;
            for (int j = i + 1; j < n; j++)
            {
                if (array[j] < array[minIndex])
                {
                    minIndex = j;
                }
            }
            // Swap array[i] and array[minIndex]
            int temp = array[i];
            array[i] = array[minIndex];
            array[minIndex] = temp;
        }
    }
}
```

Properly formatted dot string for the Selection Sort code is

```
digraph {
    start -> outer_loop_start
    outer_loop_start -> inner_loop_start
    inner_loop_start -> condition1
    condition1 -> condition2 [label="j < n"]
    condition2 -> find_min [label="array[j] < array[minIndex]"]
    find_min -> update_min
    update_min -> increment_j
    increment_j -> condition1
    condition2 -> swap
    swap -> increment_i
    increment_i -> outer_loop_start
    condition1 -> end [label="j >= n"]
    end -> print_sorted_array
    print_sorted_array -> end
}
```

The dot string response for the below Insertion Sort code is

Insertion Sort

```
using System;
class Program
{
    static void Main()
    {
        int[] array = { 64, 25, 12, 22, 11 };
```

```
        InsertionSort(array);
        Console.WriteLine("Sorted array:");
        foreach (int num in array)
        {
            Console.Write(num + " ");
        }
    }
    static void InsertionSort(int[] array)
    {
        int n = array.Length;
        for (int i = 1; i < n; i++)
        {
            int key = array[i];
            int j = i - 1;
            while (j >= 0 && array[j] > key)
            {
                array[j + 1] = array[j];
                j = j - 1;
            }
            array[j + 1] = key;
        }
    }
}
```

Properly formatted dot string for the Insertion Sort code is

```
digraph {
    start -> outer_loop_start
    outer_loop_start -> inner_loop_start
    inner_loop_start -> condition1
    condition1 -> condition2 [label="j >= 0"]
    condition2 -> move_element [label="array[j] > key"]
    move_element -> decrement_j
    decrement_j -> condition1
    condition2 -> place_key
    place_key -> increment_i
    increment_i -> outer_loop_start
    condition1 -> end [label="j < 0"]
    end -> print_sorted_array
    print_sorted_array -> end
}
```

The dot string response for the below Merge Sort code is

Merge Sort

```
using System;
class Program
{
    static void Main()
    {
        int[] array = { 64, 25, 12, 22, 11 };
        MergeSort(array, 0, array.Length - 1);
        Console.WriteLine("Sorted array:");
        foreach (int num in array)
        {
            Console.Write(num + " ");
        }
    }
    static void MergeSort(int[] array, int left, int right)
    {
        if (left < right)
```

```
        {
            int middle = left + (right - left) / 2;
            MergeSort(array, left, middle);
            MergeSort(array, middle + 1, right);
            Merge(array, left, middle, right);
        }
    }
    static void Merge(int[] array, int left, int middle, int right)
    {
        int n1 = middle - left + 1;
        int n2 = right - middle;
        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];
        Array.Copy(array, left, leftArray, 0, n1);
        Array.Copy(array, middle + 1, rightArray, 0, n2);
        int i = 0, j = 0;
        int k = left;
        while (i < n1 && j < n2)
        {
            if (leftArray[i] <= rightArray[j])
            {
                array[k] = leftArray[i];
                i++;
            }
            else
            {
                array[k] = rightArray[j];
                j++;
            }
            k++;
        }
        while (i < n1)
        {
            array[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < n2)
        {
            array[k] = rightArray[j];
            j++;
            k++;
        }
    }
}
```

Properly formatted dot string for the Merge Sort code is

```
digraph {
    start -> merge_sort
    merge_sort -> condition1
    condition1 -> merge_sort_left [label="left < right"]
    condition1 -> merge_sort_right [label="left < right"]
    merge_sort_left -> condition2
    condition2 -> merge_sort_left [label="left < right"]
    condition2 -> merge_sort_right [label="left < right"]
    merge_sort_right -> merge
    merge -> end
    end -> print_sorted_array
```

```
    print_sorted_array -> end
}
```

The dot string response for the below Quick Sort code is

Quick Sort

```
using System;
class Program
{
    static void Main()
    {
        int[] array = { 64, 25, 12, 22, 11 };
        QuickSort(array, 0, array.Length - 1);
        Console.WriteLine("Sorted array:");
        foreach (int num in array)
        {
            Console.Write(num + " ");
        }
    }
    static void QuickSort(int[] array, int low, int high)
    {
        if (low < high)
        {
            int partitionIndex = Partition(array, low, high);
            QuickSort(array, low, partitionIndex - 1);
            QuickSort(array, partitionIndex + 1, high);
        }
    }
    static int Partition(int[] array, int low, int high)
    {
        int pivot = array[high];
        int i = low - 1;
        for (int j = low; j < high; j++)
        {
            if (array[j] < pivot)
            {
                i++;
                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
        int temp1 = array[i + 1];
        array[i + 1] = array[high];
        array[high] = temp1;
        return i + 1;
    }
}
```

Properly formatted dot string for the Quick Sort code is

```
digraph {
    start -> quick_sort
    quick_sort -> condition1
    condition1 -> partition [label="low < high"]
    partition -> quick_sort_low [label="low < partitionIndex - 1"]
    partition -> quick_sort_high [label="partitionIndex + 1 < high"]
    quick_sort_low -> condition2
    condition2 -> partition [label="low < partitionIndex - 1"]
    condition2 -> quick_sort_high [label="low >= partitionIndex - 1"]
    quick_sort_high -> condition3
```

```
    condition3 -> partition [label="partitionIndex + 1 < high"]
    condition3 -> end [label="partitionIndex + 1 >= high"]
    partition -> end
    end -> print_sorted_array
    print_sorted_array -> end
}
```

The dot string response for the below Fibonacci Series code is

Fibonacci Series

```
using System;
class Program
{
    static void Main()
    {
        int n = 10;
        Console.WriteLine("Fibonacci series:");
        for (int i = 0; i < n; i++)
        {
            Console.Write(Fibonacci(i) + " ");
        }
    }
    static int Fibonacci(int n)
    {
        if (n <= 1)
        {
            return n;
        }
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

Properly formatted dot string for the Fibonacci Series code is

```
digraph {
    start -> loop_start
    loop_start -> condition
    condition -> print [label="n <= 1"]
    condition -> recursive_call [label="n > 1"]
    recursive_call -> recursive_call [label="n - 1 or n - 2"]
    recursive_call -> return [label="n <= 1"]
    print -> end
    return -> end
}
```

The dot string response for the below Palindrome Check code is

Palindrome Check

```
using System;
class Program
{
    static void Main()
    {
        string input = "level";
        if (IsPalindrome(input))
        {
            Console.WriteLine(input + " is a palindrome.");
        }
        else
        {
            Console.WriteLine(input + " is not a palindrome.");
        }
    }
```

```
    static bool IsPalindrome(string str)
    {
        int left = 0;
        int right = str.Length - 1;
        while (left < right)
        {
            if (str[left] != str[right])
            {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

Properly formatted dot string for the Palindrome Check code is

```
digraph {
    start -> condition
    condition -> compare_chars [label="left < right"]
    compare_chars -> equal_chars [label="str[left] == str[right]"]
    equal_chars -> increment_left_right
    increment_left_right -> condition
    equal_chars -> not_equal_chars [label="str[left] != str[right]"]
    not_equal_chars -> return_false
    return_false -> end
    condition -> end [label="left >= right"]
    compare_chars -> end [label="left >= right"]
    end -> end
}
```

The dot string response for the below Factorial Calculation code is

Factorial Calculation

```
using System;
class Program
{
    static void Main()
    {
        int n = 5;
        Console.WriteLine("Factorial of " + n + ": " + Factorial(n));
    }
    static int Factorial(int n)
    {
        int result = 1;
        for (int i = 1; i <= n; i++)
        {
            result *= i;
        }
        return result;
    }
}
```

Properly formatted dot string for the Factorial Calculation code is

```
digraph {
    start -> loop_start
    loop_start -> condition
    condition -> update_result [label="i <= n"]
    update_result -> update_i
    update_i -> condition
```

```
    update_result -> return_result [label="i > n"]
    return_result -> end
    end -> end
}
```

The dot string response for the below Binary Search Tree code is

Binary Search Tree

```
using System;
class TreeNode
{
    public int Value { get; set; }
    public TreeNode Left { get; set; }
    public TreeNode Right { get; set; }
    public TreeNode(int value)
    {
        Value = value;
        Left = null;
        Right = null;
    }
}
class BinarySearchTree
{
    private TreeNode root;
    public BinarySearchTree()
    {
        root = null;
    }
    public void Insert(int value)
    {
        root = InsertRecursive(root, value);
    }
    private TreeNode InsertRecursive(TreeNode node, int value)
    {
        if (node == null)
        {
            node = new TreeNode(value);
            return node;
        }
        if (value < node.Value)
        {
            node.Left = InsertRecursive(node.Left, value);
        }
        else if (value > node.Value)
        {
            node.Right = InsertRecursive(node.Right, value);
        }
        return node;
    }
    public void InOrderTraversal()
    {
        InOrderTraversalRecursive(root);
    }
    private void InOrderTraversalRecursive(TreeNode node)
    {
        if (node != null)
        {
            InOrderTraversalRecursive(node.Left);
            Console.Write(node.Value + " ");
```

```
            InOrderTraversalRecursive(node.Right);
        }
    }
}
class Program
{
    static void Main()
    {
        BinarySearchTree bst = new BinarySearchTree();
        bst.Insert(50);
        bst.Insert(30);
        bst.Insert(20);
        bst.Insert(40);
        bst.Insert(70);
        bst.Insert(60);
        bst.Insert(80);
        Console.WriteLine("In-order traversal:");
        bst.InOrderTraversal();
    }
}
```

Properly formatted dot string for the Binary Search Tree code is

```
digraph {
    start -> create_bst
    create_bst -> insert_50
    insert_50 -> insert_30 [label="value < 50"]
    insert_50 -> insert_70 [label="value > 50"]
    insert_30 -> insert_20 [label="value < 30"]
    insert_30 -> insert_40 [label="value > 30"]
    insert_70 -> insert_60 [label="value < 70"]
    insert_70 -> insert_80 [label="value > 70"]
    insert_20 -> insert_end
    insert_40 -> insert_end
    insert_60 -> insert_end
    insert_80 -> insert_end
    insert_end -> in_order_traversal
    in_order_traversal -> end
}
```

The dot string response for the below Tokenization code is

Tokenization

```
using System;
using System.Collections.Generic;
class Interpreter
{
    private readonly string input;
    private int currentTokenIndex;
    private Token currentToken;
    public Interpreter(string input)
    {
        this.input = input;
        currentTokenIndex = 0;
        currentToken = GetNextToken();
    }
    private Token GetNextToken()
    {
        if (currentTokenIndex >= input.Length)
            return new Token(TokenType.EOF, null);
        char currentChar = input[currentTokenIndex];
```

```
    currentTokenIndex++;
    if (char.IsDigit(currentChar))
    {
        return new Token(TokenType.INTEGER, int.Parse(currentChar.ToString()));
    }
    else if (currentChar == '+')
    {
        return new Token(TokenType.PLUS, currentChar);
    }
    else if (currentChar == '-')
    {
        return new Token(TokenType.MINUS, currentChar);
    }
    else if (currentChar == '*')
    {
        return new Token(TokenType.MULTIPLY, currentChar);
    }
    else if (currentChar == '/')
    {
        return new Token(TokenType.DIVIDE, currentChar);
    }
    else
    {
        throw new ArgumentException("Invalid character: " + currentChar);
    }
}
private void Eat(TokenType tokenType)
{
    if (currentToken.Type == tokenType)
    {
        currentToken = GetNextToken();
    }
    else
    {
        throw new ArgumentException("Unexpected token type: " + currentToken.Type);
    }
}
private int Factor()
{
    Token token = currentToken;
    Eat(TokenType.INTEGER);
    return (int)token.Value;
}
private int Term()
{
    int result = Factor();
    while (currentToken.Type == TokenType.MULTIPLY || currentToken.Type == TokenType.DIVIDE)
    {
        Token op = currentToken;
        if (op.Type == TokenType.MULTIPLY)
        {
            Eat(TokenType.MULTIPLY);
            result *= Factor();
        }
        else if (op.Type == TokenType.DIVIDE)
        {
            Eat(TokenType.DIVIDE);
```

```
                result /= Factor();
            }
        }
        return result;
    }
    public int Expr()
    {
        int result = Term();
        while (currentToken.Type == TokenType.PLUS || currentToken.Type == TokenType.MINUS)
        {
            Token op = currentToken;
            if (op.Type == TokenType.PLUS)
            {
                Eat(TokenType.PLUS);
                result += Term();
            }
            else if (op.Type == TokenType.MINUS)
            {
                Eat(TokenType.MINUS);
                result -= Term();
            }
        }
        return result;
    }
}
enum TokenType
{
    INTEGER,
    PLUS,
    MINUS,
    MULTIPLY,
    DIVIDE,
    EOF
}
class Token
{
    public TokenType Type { get; }
    public object Value { get; }
    public Token(TokenType type, object value)
    {
        Type = type;
        Value = value;
    }
}
class Program
{
    static void Main()
    {
        string input = "3 + 5 * 2 - 8 / 4";
        Interpreter interpreter = new Interpreter(input);
        int result = interpreter.Expr();
        Console.WriteLine("Result: " + result); // Output: 14
    }
}
```

Properly formatted dot string for the Tokenization code is

```
digraph {
    start -> interpreter_constructor
```

```
    interpreter_constructor -> get_next_token
    get_next_token -> check_eof [label="currentTokenIndex >= input.Length"]
    check_eof -> get_next_token_end [label="true"]
    check_eof -> get_next_token_continue [label="false"]
    get_next_token_continue -> check_digit [label="char.IsDigit(currentChar)"]
    check_digit -> integer_token [label="true"]
    check_digit -> check_operator [label="false"]
    integer_token -> eat_token
    check_operator -> plus_token [label="currentChar == '+'"]
    check_operator -> minus_token [label="currentChar == '-'"]
    check_operator -> multiply_token [label="currentChar == '*'"]
    check_operator -> divide_token [label="currentChar == '/'"]
    plus_token -> eat_token
    minus_token -> eat_token
    multiply_token -> eat_token
    divide_token -> eat_token
    eat_token -> get_next_token_end
    get_next_token_end -> check_eof
    get_next_token_end -> end [label="true"]
    end -> expr_start
    expr_start -> term_start
    term_start -> factor_start
    factor_start -> get_next_token_factor
    get_next_token_factor -> integer_factor [label="currentToken.Type == INTEGER"]
    integer_factor -> eat_factor
    eat_factor -> factor_end
    factor_end -> term_end
    term_end -> term_continue [label="currentToken.Type == MULTIPLY or currentToken.Type == DIVIDE"]
    term_end -> expr_continue [label="currentToken.Type != MULTIPLY and currentToken.Type != DIVIDE"]
    term_continue -> multiply_op [label="currentToken.Type == MULTIPLY"]
    term_continue -> divide_op [label="currentToken.Type == DIVIDE"]
    term_continue -> term_end [label="currentToken.Type != MULTIPLY and currentToken.Type != DIVIDE"]
    multiply_op -> eat_op
    divide_op -> eat_op
    eat_op -> factor_start
    expr_continue -> plus_op [label="currentToken.Type == PLUS"]
    expr_continue -> minus_op [label="currentToken.Type == MINUS"]
    expr_continue -> expr_end [label="currentToken.Type != PLUS and currentToken.Type != MINUS"]
    plus_op -> eat_op
    minus_op -> eat_op
    expr_end -> end [label="currentToken.Type == EOF"]
    expr_end -> expr_start [label="currentToken.Type != EOF"]
    end -> print_result
    print_result -> end
}
```

The dot string response for the below Binary Tree Insertion code is

Binary Tree Insertion

```
using System;
class TreeNode
{
    public int Value { get; set; }
    public TreeNode Left { get; set; }
    public TreeNode Right { get; set; }
    public TreeNode(int value)
    {
        Value = value;
        Left = null;
```

```
        Right = null;
    }
}
class BinaryTree
{
    private TreeNode root;
    public BinaryTree()
    {
        root = null;
    }
    public void Insert(int value)
    {
        root = InsertRecursive(root, value);
    }
    private TreeNode InsertRecursive(TreeNode node, int value)
    {
        if (node == null)
        {
            node = new TreeNode(value);
            return node;
        }
        if (value < node.Value)
        {
            node.Left = InsertRecursive(node.Left, value);
        }
        else if (value > node.Value)
        {
            node.Right = InsertRecursive(node.Right, value);
        }
        return node;
    }
}
class Program
{
    static void Main()
    {
        BinaryTree tree = new BinaryTree();
        tree.Insert(50);
        tree.Insert(30);
        tree.Insert(70);
        tree.Insert(20);
        tree.Insert(40);
        tree.Insert(60);
        tree.Insert(80);
    }
}
```

Properly formatted dot string for the Binary Tree Insertion code is

```
digraph {
    start -> create_binarytree
    create_binarytree -> insert_50
    insert_50 -> insert_30 [label="30 < 50"]
    insert_50 -> insert_70 [label="70 > 50"]
    insert_30 -> insert_20 [label="20 < 30"]
    insert_30 -> insert_40 [label="40 > 30"]
    insert_70 -> insert_60 [label="60 < 70"]
    insert_70 -> insert_80 [label="80 > 70"]
    insert_20 -> insert_end
```

```
    insert_40 -> insert_end
    insert_60 -> insert_end
    insert_80 -> insert_end
    insert_end -> end
}
```

The dot string response for the below Depth First Search code is

Depth First Search

```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }
    public void DFS(int v)
    {
        bool[] visited = new bool[V];
        DFSUtil(v, visited);
    }
    private void DFSUtil(int v, bool[] visited)
    {
        visited[v] = true;
        Console.Write(v + " ");
        foreach (int neighbor in adj[v])
        {
            if (!visited[neighbor])
            {
                DFSUtil(neighbor, visited);
            }
        }
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(4);
        graph.AddEdge(0, 1);
        graph.AddEdge(0, 2);
        graph.AddEdge(1, 2);
        graph.AddEdge(2, 0);
        graph.AddEdge(2, 3);
        graph.AddEdge(3, 3);
        Console.WriteLine("Depth First Traversal starting from vertex 2:");
        graph.DFS(2);
```

```
    }
}
```
Properly formatted dot string for the Depth First Search code is
```
digraph {
    start -> create_graph
    create_graph -> add_edge_0_1
    add_edge_0_1 -> add_edge_0_2
    add_edge_0_2 -> add_edge_1_2
    add_edge_1_2 -> add_edge_2_0
    add_edge_2_0 -> add_edge_2_3
    add_edge_2_3 -> add_edge_3_3
    add_edge_3_3 -> start_dfs
    start_dfs -> condition
    condition -> dfs_util_2 [label="!visited[2]"]
    dfs_util_2 -> condition_0 [label="!visited[0]"]
    condition_0 -> dfs_util_0 [label="!visited[0]"]
    dfs_util_0 -> condition_1 [label="!visited[1]"]
    condition_1 -> dfs_util_1 [label="!visited[1]"]
    dfs_util_1 -> condition_2 [label="!visited[2]"]
    condition_2 -> dfs_util_2 [label="!visited[2]"]
    condition_2 -> condition_3 [label="!visited[3]"]
    condition_3 -> dfs_util_3 [label="!visited[3]"]
    dfs_util_3 -> condition_3 [label="!visited[3]"]
    condition_3 -> end [label="visited all vertices"]
    end -> print_result
    print_result -> end
}
```
The dot string response for the below Breadth First Search code is
Breadth First Search
```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }
    public void BFS(int s)
    {
        bool[] visited = new bool[V];
        Queue<int> queue = new Queue<int>();
        visited[s] = true;
        queue.Enqueue(s);
        while (queue.Count != 0)
        {
            s = queue.Dequeue();
```

```
            Console.Write(s + " ");
            foreach (int neighbor in adj[s])
            {
                if (!visited[neighbor])
                {
                    visited[neighbor] = true;
                    queue.Enqueue(neighbor);
                }
            }
        }
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(4);
        graph.AddEdge(0, 1);
        graph.AddEdge(0, 2);
        graph.AddEdge(1, 2);
        graph.AddEdge(2, 0);
        graph.AddEdge(2, 3);
        graph.AddEdge(3, 3);
        Console.WriteLine("Breadth First Traversal starting from vertex 2:");
        graph.BFS(2);
    }
}
```

Properly formatted dot string for the Breadth First Search code is

```
digraph {
    start -> create_graph
    create_graph -> add_edge_0_1
    add_edge_0_1 -> add_edge_0_2
    add_edge_0_2 -> add_edge_1_2
    add_edge_1_2 -> add_edge_2_0
    add_edge_2_0 -> add_edge_2_3
    add_edge_2_3 -> add_edge_3_3
    add_edge_3_3 -> start_bfs
    start_bfs -> condition
    condition -> print_start_vertex
    print_start_vertex -> enqueue_start_vertex
    enqueue_start_vertex -> loop_start
    loop_start -> dequeue
    dequeue -> condition1
    condition1 -> print_current_vertex
    print_current_vertex -> condition2 [label="!visited[neighbor]"]
    print_current_vertex -> loop_start [label="visited[neighbor]"]
    condition2 -> visit_neighbor [label="!visited[neighbor]"]
    visit_neighbor -> enqueue_neighbor
    enqueue_neighbor -> condition3
    condition3 -> loop_start
    condition3 -> end [label="queue is empty"]
    end -> print_result
    print_result -> end
}
```

The dot string response for the below Topological Sorting in Directed Acyclic Graph code is

Topological Sorting in Directed Acyclic Graph

```
using System;
```

```
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }
    public void TopologicalSort()
    {
        Stack<int> stack = new Stack<int>();
        bool[] visited = new bool[V];
        for (int i = 0; i < V; i++)
        {
            if (!visited[i])
            {
                TopologicalSortUtil(i, visited, stack);
            }
        }
        Console.WriteLine("Topological Sorting:");
        while (stack.Count != 0)
        {
            Console.Write(stack.Pop() + " ");
        }
    }
    private void TopologicalSortUtil(int v, bool[] visited, Stack<int> stack)
    {
        visited[v] = true;
        foreach (int neighbor in adj[v])
        {
            if (!visited[neighbor])
            {
                TopologicalSortUtil(neighbor, visited, stack);
            }
        }
        stack.Push(v);
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(6);
        graph.AddEdge(5, 2);
        graph.AddEdge(5, 0);
        graph.AddEdge(4, 0);
        graph.AddEdge(4, 1);
```

```
        graph.AddEdge(2, 3);
        graph.AddEdge(3, 1);
        graph.TopologicalSort();
    }
}
```

Properly formatted dot string for the Topological Sorting in Directed Acyclic Graph code is

```
digraph {
    start -> create_graph
    create_graph -> add_edge_5_2
    add_edge_5_2 -> add_edge_5_0
    add_edge_5_0 -> add_edge_4_0
    add_edge_4_0 -> add_edge_4_1
    add_edge_4_1 -> add_edge_2_3
    add_edge_2_3 -> add_edge_3_1
    add_edge_3_1 -> start_topological_sort
    start_topological_sort -> loop_start
    loop_start -> condition
    condition -> visit_vertex [label="!visited[i]"]
    visit_vertex -> dfs_util
    dfs_util -> loop_start
    condition -> end [label="visited all vertices"]
    end -> print_result
    print_result -> end
}
```

The dot string response for the below Dijkstras Algorithm code is

Dijkstras Algorithm

```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<Tuple<int, int>>[] adj;
    public Graph(int v)
    {
        V = v;
        adj = new List<Tuple<int, int>>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<Tuple<int, int>>();
        }
    }
    public void AddEdge(int u, int v, int weight)
    {
        adj[u].Add(new Tuple<int, int>(v, weight));
        adj[v].Add(new Tuple<int, int>(u, weight)); // For undirected graph
    }
    public void Dijkstra(int src)
    {
        int[] dist = new int[V];
        bool[] sptSet = new bool[V];
        for (int i = 0; i < V; i++)
        {
            dist[i] = int.MaxValue;
        }
        dist[src] = 0;
        for (int count = 0; count < V - 1; count++)
        {
```

```
            int u = MinDistance(dist, sptSet);
            sptSet[u] = true;
            foreach (var neighbor in adj[u])
            {
                int v = neighbor.Item1;
                int weight = neighbor.Item2;
                if (!sptSet[v] && dist[u] != int.MaxValue && dist[u] + weight < dist[v])
                {
                    dist[v] = dist[u] + weight;
                }
            }
        }
        Console.WriteLine("Vertex   Distance from Source");
        for (int i = 0; i < V; i++)
        {
            Console.WriteLine(i + "\t" + dist[i]);
        }
    }
    private int MinDistance(int[] dist, bool[] sptSet)
    {
        int min = int.MaxValue, minIndex = -1;
        for (int v = 0; v < V; v++)
        {
            if (sptSet[v] == false && dist[v] <= min)
            {
                min = dist[v];
                minIndex = v;
            }
        }
        return minIndex;
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(9);
        graph.AddEdge(0, 1, 4);
        graph.AddEdge(0, 7, 8);
        graph.AddEdge(1, 2, 8);
        graph.AddEdge(1, 7, 11);
        graph.AddEdge(2, 3, 7);
        graph.AddEdge(2, 8, 2);
        graph.AddEdge(2, 5, 4);
        graph.AddEdge(3, 4, 9);
        graph.AddEdge(3, 5, 14);
        graph.AddEdge(4, 5, 10);
        graph.AddEdge(5, 6, 2);
        graph.AddEdge(6, 7, 1);
        graph.AddEdge(6, 8, 6);
        graph.AddEdge(7, 8, 7);
        graph.Dijkstra(0);
    }
}
```

Properly formatted dot string for the Dijkstras Algorithm code is

```
digraph {
    start -> create_graph
```

```
    create_graph -> add_edge_0_1
    add_edge_0_1 -> add_edge_0_7
    add_edge_0_7 -> add_edge_1_2
    add_edge_1_2 -> add_edge_1_7
    add_edge_1_7 -> add_edge_2_3
    add_edge_2_3 -> add_edge_2_8
    add_edge_2_8 -> add_edge_2_5
    add_edge_2_5 -> add_edge_3_4
    add_edge_3_4 -> add_edge_3_5
    add_edge_3_5 -> add_edge_4_5
    add_edge_4_5 -> add_edge_5_6
    add_edge_5_6 -> add_edge_6_7
    add_edge_6_7 -> add_edge_6_8
    add_edge_6_8 -> add_edge_7_8
    add_edge_7_8 -> start_dijkstra
    start_dijkstra -> initialize_distances
    initialize_distances -> loop_start
    loop_start -> min_distance
    min_distance -> set_spt_set_true
    set_spt_set_true -> loop_start
    loop_start -> condition1
    condition1 -> update_distances
    update_distances -> condition2
    condition2 -> loop_start
    condition2 -> end [label="V - 1 iterations completed"]
    end -> print_result
    print_result -> end
}
```

The dot string response for the below Bellman Ford Algorithm code is

Bellman Ford Algorithm

```
using System;
using System.Collections.Generic;
class Graph
{
    class Edge
    {
        public int Source, Destination, Weight;
        public Edge()
        {
            Source = Destination = Weight = 0;
        }
    }
    private int V, E;
    private Edge[] edges;
    public Graph(int v, int e)
    {
        V = v;
        E = e;
        edges = new Edge[e];
        for (int i = 0; i < e; ++i)
        {
            edges[i] = new Edge();
        }
    }
    public void AddEdge(int u, int v, int w, int index)
    {
        edges[index].Source = u;
```

```
        edges[index].Destination = v;
        edges[index].Weight = w;
    }
    public void BellmanFord(int src)
    {
        int[] dist = new int[V];
        for (int i = 0; i < V; ++i)
        {
            dist[i] = int.MaxValue;
        }
        dist[src] = 0;
        for (int i = 1; i < V; ++i)
        {
            for (int j = 0; j < E; ++j)
            {
                int u = edges[j].Source;
                int v = edges[j].Destination;
                int weight = edges[j].Weight;
                if (dist[u] != int.MaxValue && dist[u] + weight < dist[v])
                {
                    dist[v] = dist[u] + weight;
                }
            }
        }
        Console.WriteLine("Vertex   Distance from Source");
        for (int i = 0; i < V; ++i)
        {
            Console.WriteLine(i + "\t\t" + dist[i]);
        }
    }
}
class Program
{
    static void Main()
    {
        int V = 5; // Number of vertices in graph
        int E = 8; // Number of edges in graph
        Graph graph = new Graph(V, E);
        // Add edge 0-1
        graph.AddEdge(0, 1, -1, 0);
        // Add edge 0-2
        graph.AddEdge(0, 2, 4, 1);
        // Add edge 1-2
        graph.AddEdge(1, 2, 3, 2);
        // Add edge 1-3
        graph.AddEdge(1, 3, 2, 3);
        // Add edge 1-4
        graph.AddEdge(1, 4, 2, 4);
        // Add edge 3-2
        graph.AddEdge(3, 2, 5, 5);
        // Add edge 3-1
        graph.AddEdge(3, 1, 1, 6);
        // Add edge 4-3
        graph.AddEdge(4, 3, -3, 7);
        graph.BellmanFord(0);
    }
}
```

Properly formatted dot string for the Bellman Ford Algorithm code is

```
digraph {
    start -> create_graph
    create_graph -> add_edge_0_1
    add_edge_0_1 -> add_edge_0_2
    add_edge_0_2 -> add_edge_1_2
    add_edge_1_2 -> add_edge_1_3
    add_edge_1_3 -> add_edge_1_4
    add_edge_1_4 -> add_edge_3_2
    add_edge_3_2 -> add_edge_3_1
    add_edge_3_1 -> add_edge_4_3
    add_edge_4_3 -> start_bellman_ford
    start_bellman_ford -> initialize_distances
    initialize_distances -> loop_start
    loop_start -> update_distances
    update_distances -> loop_start
    loop_start -> print_result
    print_result -> end
}
```

The dot string response for the below Floyd Warshall code is

Floyd Warshall

```
using System;
class Graph
{
    private int V;
    private int[,] dist;
    public Graph(int v)
    {
        V = v;
        dist = new int[V, V];
        for (int i = 0; i < V; ++i)
        {
            for (int j = 0; j < V; ++j)
            {
                dist[i, j] = int.MaxValue;
            }
        }
    }
    public void AddEdge(int u, int v, int weight)
    {
        dist[u, v] = weight;
    }
    public void FloydWarshall()
    {
        int[,] distCopy = new int[V, V];
        Array.Copy(dist, distCopy, V * V);
        for (int k = 0; k < V; ++k)
        {
            for (int i = 0; i < V; ++i)
            {
                for (int j = 0; j < V; ++j)
                {
                    if (distCopy[i, k] != int.MaxValue && distCopy[k, j] != int.MaxValue &&
                        distCopy[i, k] + distCopy[k, j] < distCopy[i, j])
                    {
                        distCopy[i, j] = distCopy[i, k] + distCopy[k, j];
                    }
```

```
            }
        }
    }
    Console.WriteLine("Shortest distances between every pair of vertices:");
    for (int i = 0; i < V; ++i)
    {
        for (int j = 0; j < V; ++j)
        {
            if (distCopy[i, j] == int.MaxValue)
            {
                Console.Write("INF\t");
            }
            else
            {
                Console.Write(distCopy[i, j] + "\t");
            }
        }
        Console.WriteLine();
    }
  }
}
class Program
{
  static void Main()
  {
    int V = 4;
    Graph graph = new Graph(V);
    graph.AddEdge(0, 1, 5);
    graph.AddEdge(0, 3, 10);
    graph.AddEdge(1, 2, 3);
    graph.AddEdge(2, 3, 1);
    graph.FloydWarshall();
  }
}
```

Properly formatted dot string for the Floyd Warshall code is

```
digraph {
  start -> create_graph
  create_graph -> add_edge_0_1
  add_edge_0_1 -> add_edge_0_3
  add_edge_0_3 -> add_edge_1_2
  add_edge_1_2 -> add_edge_2_3
  add_edge_2_3 -> start_floyd_warshall
  start_floyd_warshall -> loop_k
  loop_k -> loop_i
  loop_i -> loop_j
  loop_j -> condition1
  condition1 -> update_distance [label="distCopy[i, k] != int.MaxValue && distCopy[k, j] != int.MaxValue && distCopy[i, k] +
distCopy[k, j] < distCopy[i, j]"]
  update_distance -> loop_j
  loop_j -> condition2
  condition2 -> loop_i
  loop_i -> condition3
  condition3 -> loop_k
  condition3 -> print_result [label="k = V - 1"]
  print_result -> end
}
```

The dot string response for the below Kruskal Algorithm code is

Kruskal Algorithm

```
using System;
using System.Collections.Generic;
class Graph
{
    class Edge : IComparable<Edge>
    {
        public int Source, Destination, Weight;
        public int CompareTo(Edge other)
        {
            return this.Weight - other.Weight;
        }
    }
    private int V;
    private List<Edge> edges;
    public Graph(int v)
    {
        V = v;
        edges = new List<Edge>();
    }
    public void AddEdge(int u, int v, int w)
    {
        edges.Add(new Edge { Source = u, Destination = v, Weight = w });
    }
    private int Find(int[] parent, int i)
    {
        if (parent[i] == -1)
        {
            return i;
        }
        return Find(parent, parent[i]);
    }
    private void Union(int[] parent, int x, int y)
    {
        int xSet = Find(parent, x);
        int ySet = Find(parent, y);
        parent[xSet] = ySet;
    }
    public void KruskalMST()
    {
        List<Edge> result = new List<Edge>();
        int[] parent = new int[V];
        for (int i = 0; i < V; ++i)
        {
            parent[i] = -1;
        }
        edges.Sort();
        int e = 0;
        int i = 0;
        while (e < V - 1 && i < edges.Count)
        {
            Edge nextEdge = edges[i++];
            int x = Find(parent, nextEdge.Source);
            int y = Find(parent, nextEdge.Destination);
            if (x != y)
            {
                result.Add(nextEdge);
```

```
            Union(parent, x, y);
            e++;
        }
    }
    Console.WriteLine("Edges in the Minimum Spanning Tree:");
    foreach (var edge in result)
    {
        Console.WriteLine($"{edge.Source} - {edge.Destination} : {edge.Weight}");
    }
  }
}
class Program
{
  static void Main()
  {
    int V = 4;
    Graph graph = new Graph(V);
    graph.AddEdge(0, 1, 10);
    graph.AddEdge(0, 2, 6);
    graph.AddEdge(0, 3, 5);
    graph.AddEdge(1, 3, 15);
    graph.AddEdge(2, 3, 4);
    graph.KruskalMST();
  }
}
```

Properly formatted dot string for the Kruskal Algorithm code is

```
digraph {
  start -> create_graph
  create_graph -> add_edge_0_1
  add_edge_0_1 -> add_edge_0_2
  add_edge_0_2 -> add_edge_0_3
  add_edge_0_3 -> add_edge_1_3
  add_edge_1_3 -> start_kruskal_mst
  start_kruskal_mst -> sort_edges
  sort_edges -> loop_start
  loop_start -> condition [label="e < V - 1 && i < edges.Count"]
  condition -> find_source_set [label="Find(parent, nextEdge.Source)"]
  find_source_set -> find_destination_set [label="Find(parent, nextEdge.Destination)"]
  find_destination_set -> condition1 [label="x != y"]
  condition1 -> add_edge_to_mst [label="result.Add(nextEdge)"]
  add_edge_to_mst -> union_sets [label="Union(parent, x, y)"]
  union_sets -> increment_e [label="e++"]
  increment_e -> loop_start
  condition -> end [label="e == V - 1 || i == edges.Count"]
  end -> print_result
  print_result -> end
}
```

The dot string response for the below Prim’s Algorithm code is

Prim’s Algorithm

```
using System;
using System.Collections.Generic;
class Graph
{
  private int V;
  private List<List<Tuple<int, int>>> adj;
  public Graph(int v)
  {
```

```
    V = v;
    adj = new List<List<Tuple<int, int>>>(v);
    for (int i = 0; i < v; ++i)
    {
        adj.Add(new List<Tuple<int, int>>());
    }
}
public void AddEdge(int u, int v, int w)
{
    adj[u].Add(new Tuple<int, int>(v, w));
    adj[v].Add(new Tuple<int, int>(u, w)); // For undirected graph
}
public void PrimMST()
{
    bool[] inMST = new bool[V];
    int[] key = new int[V];
    int[] parent = new int[V];
    for (int i = 0; i < V; ++i)
    {
        key[i] = int.MaxValue;
        inMST[i] = false;
    }
    key[0] = 0; // Start from vertex 0
    for (int count = 0; count < V - 1; ++count)
    {
        int u = MinKey(key, inMST);
        inMST[u] = true;
        foreach (var node in adj[u])
        {
            int v = node.Item1;
            int weight = node.Item2;
            if (!inMST[v] && weight < key[v])
            {
                parent[v] = u;
                key[v] = weight;
            }
        }
    }
    Console.WriteLine("Edges in the Minimum Spanning Tree (Prim's Algorithm):");
    for (int i = 1; i < V; ++i)
    {
        Console.WriteLine(parent[i] + " - " + i + "\t: " + key[i]);
    }
}
private int MinKey(int[] key, bool[] inMST)
{
    int min = int.MaxValue, minIndex = -1;
    for (int v = 0; v < V; ++v)
    {
        if (!inMST[v] && key[v] < min)
        {
            min = key[v];
            minIndex = v;
        }
    }
    return minIndex;
}
```

```
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(5);
        graph.AddEdge(0, 1, 2);
        graph.AddEdge(0, 3, 6);
        graph.AddEdge(1, 2, 3);
        graph.AddEdge(1, 3, 8);
        graph.AddEdge(1, 4, 5);
        graph.AddEdge(2, 4, 7);
        graph.AddEdge(3, 4, 9);
        graph.PrimMST();
    }
}
```

Properly formatted dot string for the Prim's Algorithm code is

```
digraph {
    start -> create_graph
    create_graph -> add_edge_0_1
    add_edge_0_1 -> add_edge_0_3
    add_edge_0_3 -> add_edge_1_2
    add_edge_1_2 -> add_edge_1_3
    add_edge_1_3 -> add_edge_1_4
    add_edge_1_4 -> add_edge_2_4
    add_edge_2_4 -> add_edge_3_4
    add_edge_3_4 -> start_prim_mst
    start_prim_mst -> initialize_data_structures
    initialize_data_structures -> loop_start
    loop_start -> min_key
    min_key -> set_in_mst_true
    set_in_mst_true -> loop_start
    loop_start -> condition1 [label="count < V - 1"]
    condition1 -> find_min_key [label="MinKey(key, inMST)"]
    find_min_key -> set_in_mst_true1
    set_in_mst_true1 -> loop_start
    condition1 -> print_result [label="count == V - 1"]
    print_result -> end
}
```

The dot string response for the below Bipartite Graph Detection using Breadth-First Search code is

Bipartite Graph Detection using Breadth-First Search

```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
```

```
    {
        adj[v].Add(w);
        adj[w].Add(v); // For undirected graph
    }
    public bool IsBipartite()
    {
        int[] colors = new int[V];
        for (int i = 0; i < V; ++i)
        {
            colors[i] = -1;
        }
        for (int i = 0; i < V; ++i)
        {
            if (colors[i] == -1)
            {
                if (!BFSUtil(i, colors))
                {
                    return false;
                }
            }
        }
        return true;
    }
    private bool BFSUtil(int src, int[] colors)
    {
        Queue<int> queue = new Queue<int>();
        queue.Enqueue(src);
        colors[src] = 1;
        while (queue.Count != 0)
        {
            int u = queue.Dequeue();
            foreach (int v in adj[u])
            {
                if (colors[v] == -1)
                {
                    colors[v] = 1 - colors[u];
                    queue.Enqueue(v);
                }
                else if (colors[v] == colors[u])
                {
                    return false;
                }
            }
        }
        return true;
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(4);
        graph.AddEdge(0, 1);
        graph.AddEdge(0, 3);
        graph.AddEdge(1, 2);
        graph.AddEdge(2, 3);
        Console.WriteLine("Is the graph bipartite? " + graph.IsBipartite());
```

```
    }
}
```
Properly formatted dot string for the Bipartite Graph Detection using Breadth-First Search code is
```
digraph {
    start -> create_graph
    create_graph -> add_edge_0_1
    add_edge_0_1 -> add_edge_0_3
    add_edge_0_3 -> add_edge_1_2
    add_edge_1_2 -> add_edge_2_3
    add_edge_2_3 -> start_is_bipartite
    start_is_bipartite -> initialize_colors
    initialize_colors -> loop_start
    loop_start -> condition
    condition -> set_color_u [label="colors[i] == -1"]
    set_color_u -> loop_start
    condition -> bfs_util [label="colors[i] != -1"]
    bfs_util -> condition1
    condition1 -> check_bipartite [label="colors[v] == -1"]
    check_bipartite -> set_color_v
    set_color_v -> enqueue_v
    enqueue_v -> loop_start
    check_bipartite -> condition2 [label="colors[v] != -1"]
    condition2 -> check_colors [label="colors[v] == colors[u]"]
    check_colors -> end [label="Not Bipartite"]
    check_colors -> loop_start [label="Bipartite"]
    end -> print_result
    print_result -> end
}
```
The dot string response for the below Articulation Points code is
Articulation Points
```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    private int time;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
        adj[w].Add(v); // For undirected graph
    }
    public void FindArticulationPoints()
    {
        bool[] visited = new bool[V];
        int[] parent = new int[V];
        int[] disc = new int[V];
        int[] low = new int[V];
```

```
        bool[] ap = new bool[V];
        for (int i = 0; i < V; ++i)
        {
            parent[i] = -1;
            visited[i] = false;
            ap[i] = false;
        }
        for (int i = 0; i < V; ++i)
        {
            if (!visited[i])
            {
                APUtil(i, visited, parent, disc, low, ap);
            }
        }
        Console.WriteLine("Articulation Points in the Graph:");
        for (int i = 0; i < V; ++i)
        {
            if (ap[i])
            {
                Console.Write(i + " ");
            }
        }
    }
    private void APUtil(int u, bool[] visited, int[] parent, int[] disc, int[] low, bool[] ap)
    {
        int children = 0;
        visited[u] = true;
        disc[u] = low[u] = ++time;
        foreach (int v in adj[u])
        {
            if (!visited[v])
            {
                children++;
                parent[v] = u;
                APUtil(v, visited, parent, disc, low, ap);
                low[u] = Math.Min(low[u], low[v]);
                if (parent[u] == -1 && children > 1)
                {
                    ap[u] = true;
                }
                if (parent[u] != -1 && low[v] >= disc[u])
                {
                    ap[u] = true;
                }
            }
            else if (v != parent[u])
            {
                low[u] = Math.Min(low[u], disc[v]);
            }
        }
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(5);
```

```
        graph.AddEdge(1, 0);
        graph.AddEdge(0, 2);
        graph.AddEdge(2, 1);
        graph.AddEdge(0, 3);
        graph.AddEdge(3, 4);
        graph.FindArticulationPoints();
    }
}
```

Properly formatted dot string for the Articulation Points code is

```
digraph {
    start -> create_graph
    create_graph -> add_edge_1_0
    add_edge_1_0 -> add_edge_0_2
    add_edge_0_2 -> add_edge_2_1
    add_edge_2_1 -> add_edge_0_3
    add_edge_0_3 -> add_edge_3_4
    add_edge_3_4 -> start_find_articulation_points
    start_find_articulation_points -> initialize_data_structures
    initialize_data_structures -> loop_start
    loop_start -> condition
    condition -> ap_util
    ap_util -> condition1
    condition1 -> loop_start [label="v != parent[u]"]
    condition1 -> set_low_u [label="low[u] = Math.Min(low[u], disc[v])"]
    set_low_u -> loop_start
    condition -> condition2 [label="parent[u] == -1 && children > 1"]
    condition2 -> set_ap_u [label="ap[u] = true"]
    set_ap_u -> loop_start
    condition -> condition3 [label="parent[u] != -1 && low[v] >= disc[u]"]
    condition3 -> set_ap_u1 [label="ap[u] = true"]
    set_ap_u1 -> loop_start
    condition -> print_result [label="Visited all vertices"]
    print_result -> end
    end -> end
}
```

The dot string response for the kahn’s Algorithm below code is

kahn’s Algorithm

```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }
    public void TopologicalSort()
```

```
    {
        int[] inDegree = new int[V];
        foreach (var neighbors in adj)
        {
            foreach (var neighbor in neighbors)
            {
                inDegree[neighbor]++;
            }
        }
        Queue<int> queue = new Queue<int>();
        for (int i = 0; i < V; ++i)
        {
            if (inDegree[i] == 0)
            {
                queue.Enqueue(i);
            }
        }
        int count = 0;
        List<int> topologicalOrder = new List<int>();
        while (queue.Count != 0)
        {
            int u = queue.Dequeue();
            topologicalOrder.Add(u);
            foreach (int neighbor in adj[u])
            {
                if (--inDegree[neighbor] == 0)
                {
                    queue.Enqueue(neighbor);
                }
            }
            count++;
        }
        if (count != V)
        {
            Console.WriteLine("Graph contains cycle. Topological sorting is not possible.");
            return;
        }
        Console.WriteLine("Topological Sorting (Kahn's Algorithm):");
        foreach (var vertex in topologicalOrder)
        {
            Console.Write(vertex + " ");
        }
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(6);
        graph.AddEdge(5, 2);
        graph.AddEdge(5, 0);
        graph.AddEdge(4, 0);
        graph.AddEdge(4, 1);
        graph.AddEdge(2, 3);
        graph.AddEdge(3, 1);
        graph.TopologicalSort();
    }
```

```
}
```
Properly formatted dot string for the kahn's Algorithm code is
```
digraph {
    start -> create_graph
    create_graph -> add_edge_5_2
    add_edge_5_2 -> add_edge_5_0
    add_edge_5_0 -> add_edge_4_0
    add_edge_4_0 -> add_edge_4_1
    add_edge_4_1 -> add_edge_2_3
    add_edge_2_3 -> add_edge_3_1
    add_edge_3_1 -> start_topological_sort
    start_topological_sort -> initialize_indegree
    initialize_indegree -> loop_start
    loop_start -> condition
    condition -> enqueue_vertex [label="inDegree[i] == 0"]
    enqueue_vertex -> loop_start
    condition -> increment_count
    increment_count -> loop_start
    condition -> check_count [label="count != V"]
    check_count -> print_result [label="Graph contains cycle"]
    check_count -> end [label="Topological sorting possible"]
    print_result -> end
    end -> end
}
```
The dot string response for the below Kosaraju's Algorithm code is
Kosaraju's Algorithm
```
using System;
using System.Collections.Generic;
class Graph
{
    private int V;
    private List<int>[] adj;
    private Stack<int> stack;
    public Graph(int v)
    {
        V = v;
        adj = new List<int>[v];
        for (int i = 0; i < v; ++i)
        {
            adj[i] = new List<int>();
        }
        stack = new Stack<int>();
    }
    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }
    public List<List<int>> GetSCCs()
    {
        bool[] visited = new bool[V];
        for (int i = 0; i < V; ++i)
        {
            if (!visited[i])
            {
                DFSUtil(i, visited);
            }
        }
```

```
        Graph transposedGraph = Transpose();
        List<List<int>> sccs = new List<List<int>>();
        visited = new bool[V];
        while (stack.Count != 0)
        {
            int v = stack.Pop();
            if (!visited[v])
            {
                List<int> scc = new List<int>();
                transposedGraph.DFSUtil(v, visited, scc);
                sccs.Add(scc);
            }
        }
        return sccs;
    }
    private void DFSUtil(int v, bool[] visited, List<int> scc)
    {
        visited[v] = true;
        scc.Add(v);
        foreach (int neighbor in adj[v])
        {
            if (!visited[neighbor])
            {
                DFSUtil(neighbor, visited, scc);
            }
        }
    }
    private Graph Transpose()
    {
        Graph transposed = new Graph(V);
        for (int v = 0; v < V; ++v)
        {
            foreach (int neighbor in adj[v])
            {
                transposed.AddEdge(neighbor, v);
            }
        }
        return transposed;
    }
    private void FillOrder(int v, bool[] visited)
    {
        visited[v] = true;
        foreach (int neighbor in adj[v])
        {
            if (!visited[neighbor])
            {
                FillOrder(neighbor, visited);
            }
        }
        stack.Push(v);
    }
}
class Program
{
    static void Main()
    {
        Graph graph = new Graph(5);
```

```
        graph.AddEdge(1, 0);
        graph.AddEdge(0, 2);
        graph.AddEdge(2, 1);
        graph.AddEdge(0, 3);
        graph.AddEdge(3, 4);
        List<List<int>> sccs = graph.GetSCCs();
        Console.WriteLine("Strongly Connected Components:");
        foreach (var scc in sccs)
        {
            Console.WriteLine(string.Join(" ", scc));
        }
    }
}
```

Properly formatted dot string for the Kosaraju's Algorithm code is

```
digraph {
    start -> create_graph
    create_graph -> add_edge_1_0
    add_edge_1_0 -> add_edge_0_2
    add_edge_0_2 -> add_edge_2_1
    add_edge_2_1 -> add_edge_0_3
    add_edge_0_3 -> add_edge_3_4
    add_edge_3_4 -> start_get_sccs
    start_get_sccs -> initialize_data_structures
    initialize_data_structures -> loop_start
    loop_start -> condition
    condition -> dfs_util
    dfs_util -> loop_start
    condition -> fill_order
    fill_order -> loop_start
    condition -> transpose_graph
    transpose_graph -> start_get_sccs1
    start_get_sccs1 -> initialize_data_structures1
    initialize_data_structures1 -> loop_start1
    loop_start1 -> condition1
    condition1 -> dfs_util1
    dfs_util1 -> loop_start1
    condition1 -> fill_order1
    fill_order1 -> loop_start1
    condition1 -> print_result [label="Stack is not empty"]
    print_result -> end
    end -> end
}
```

The dot string response for the below Prime number check code is

Prime number check

```
using System;
public class PrimeChecker
{
    public static bool IsPrime(int num)
    {
        if (num <= 1)
        {
            return false;
        }
        for (int i = 2; i <= Math.Sqrt(num); i++)
        {
            if (num % i == 0)
            {
```

```
            return false;
        }
    }
    return true;
  }
}
class Program
{
  static void Main()
  {
    int number = 17;
    if (PrimeChecker.IsPrime(number))
    {
      Console.WriteLine(number + " is a prime number.");
    }
    else
    {
      Console.WriteLine(number + " is not a prime number.");
    }
  }
}
```

Properly formatted dot string for the Prime number check code is

```
digraph {
  start -> check_prime
  check_prime -> condition1 [label="num <= 1"]
  condition1 -> print_not_prime [label="True"]
  print_not_prime -> end
  condition1 -> loop_start [label="False"]
  loop_start -> condition2 [label="i <= √num"]
  condition2 -> condition3 [label="num % i == 0"]
  condition3 -> print_not_prime [label="True"]
  print_not_prime -> end
  condition3 -> loop_increment [label="False"]
  loop_increment -> loop_start
  condition2 -> print_prime [label="False"]
  print_prime -> end
  end -> end
}
```

The dot string response for the below Merging Two Dictionaries C# code is

```
Merging Two Dictionaries
Dictionary<string, string> dict1 = new Dictionary<string, string> { { "Superman", "Flight" } };
Dictionary<string, string> dict2 = new Dictionary<string, string> { { "Batman", "Gadgets" } };
// Using LINQ
var merged = dict1.Concat(dict2).ToDictionary(x => x.Key, x => x.Value);
// Using a foreach loop
foreach (var item in dict2)
{
  dict1[item.Key] = item.Value;
}
// Using the Union extension method
var merged2 = dict1.Union(dict2).ToDictionary(x => x.Key, x => x.Value);
```

Properly formatted dot string for the Above Merging Two Dictionaries code

```
digraph ControlFlowDiagram {
  start [shape=ellipse, label="Start"];
  dict1 [shape=box, label="Dictionary<string, string> dict1 = new Dictionary<string, string> { { \"Superman\", \"Flight\" } };"];
  dict2 [shape=box, label="Dictionary<string, string> dict2 = new Dictionary<string, string> { { \"Batman\", \"Gadgets\" } };"];
  merged [shape=box, label="var merged = dict1.Concat(dict2).ToDictionary(x => x.Key, x => x.Value);"];
```

```
    foreach [shape=box, label="foreach (var item in dict2)\n{\n    dict1[item.Key] = item.Value;\n}"];
    merged2 [shape=box, label="var merged2 = dict1.Union(dict2).ToDictionary(x => x.Key, x => x.Value);"];
    end [shape=ellipse, label="End"];
    start -> dict1;
    dict1 -> merged [label="Concat"];
    dict2 -> merged;
    merged -> foreach [label="LINQ"];
    foreach -> merged2 [label="Loop"];
    merged -> merged2 [label="Union"];
    merged2 -> end;
}
```