EAST WEST UNIVERSITY

Lab 6 Report

Lab 6 Topic: Generate a graph to find the Cyclomatic complexity of a C Code.

Course Title: Software Engineering

Course Code: CSE412

Section: 03

Submitted to:

Nishat Tasnim Niloy
Lecturer
Department of Computer Science & Engineering,
East West University

Prepared by:

Group Member Name
Golam Kibria (2021-3-60-215)
B. M. Shahria Alam (2021-3-60-016)
A.B.M. Ilman Farabi (2021-3-60-111)
Shamima Sharmin Ananna (2022-1-60-148)

Date of submission: 14/01/2025

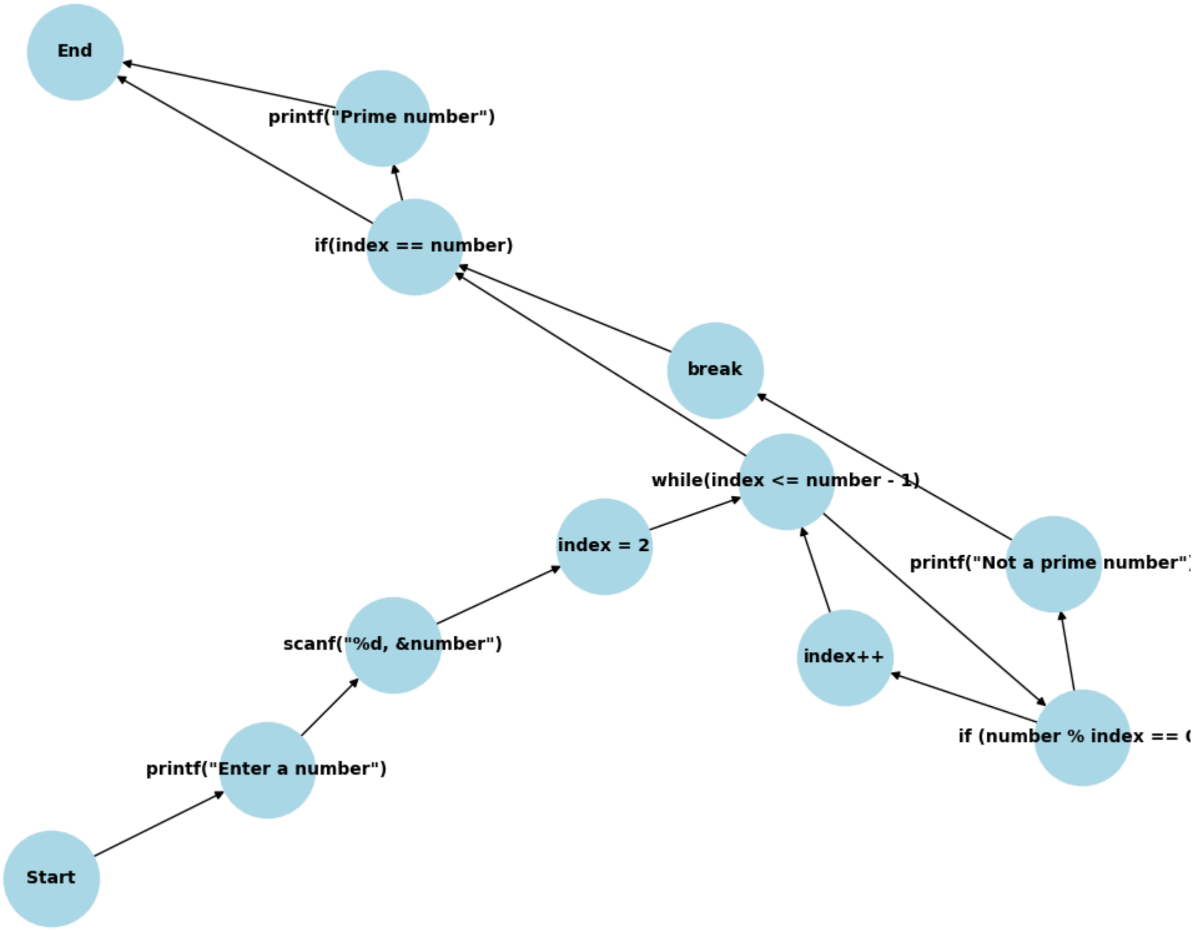
Parse the Code:

Analyze the code structure to identify control flow elements like decisions (if, while, break).

```
import networkx as nx
import matplotlib.pyplot as plt

def generate_cfg():
    # Define nodes and edges based on the code structure
    G = nx.DiGraph()

    # Nodes represent the statements
    nodes = {
        1: "Start",
        2: "printf(\"Enter a number\")",
        3: "scanf(\"%d, &number\")",
        4: "index = 2",
        5: "while(index <= number - 1)",
        6: "if (number % index == 0)",
        7: "printf(\"Not a prime number\")",
        8: "break",
        9: "index++",
        10: "if(index == number)",
        11: "printf(\"Prime number\")",
        12: "End"
    }

    # Add edges to represent control flow
    edges = [
        (1, 2),
        (2, 3),
        (3, 4),
        (4, 5),
        (5, 6),
        (6, 7),
        (6, 9),
        (7, 8),
        (8, 10),
        (9, 5),
        (5, 10),
        (10, 11),
        (10, 12),
        (11, 12)
    ]

    # Add nodes and edges to the graph
    G.add_nodes_from(nodes.keys())
    G.add_edges_from(edges)

    return G, nodes

def calculate_cyclomatic_complexity(G):
    # Cyclomatic Complexity = E - N + 2P
    edges = G.number_of_edges()
    nodes = G.number_of_nodes()
    connected_components = nx.number_weakly_connected_components(G)

    complexity = edges - nodes + 2 * connected_components
    return complexity

def visualize_cfg(G, nodes):
    pos = nx.spring_layout(G)
    labels = {node: nodes[node] for node in G.nodes()}

    plt.figure(figsize=(10, 8))
    nx.draw(G, pos, with_labels=True, labels=labels, node_size=3000, node_color='lightblue',
font_size=10, font_weight='bold')
    plt.title("Control Flow Graph")
    plt.show()

if __name__ == "__main__":
    G, nodes = generate_cfg()
    complexity = calculate_cyclomatic_complexity(G)

    print("Cyclomatic Complexity:", complexity)
    visualize_cfg(G, nodes)
```

Generate a Control Flow Graph (CFG):

- □ Represent each statement as a node.
- □ Represent control flow changes (e.g., from conditions) as edges.



Calculate Cyclomatic Complexity:

- Cyclomatic Complexity = E - N + 2P
  - E: Number of edges=10.
  - N: Number of nodes=8.
  - P: Number of connected components (typically 1 for a single program)=1.

Cyclomatic Complexity = 14 - 12 + 2

Cyclomatic Complexity: 4

Test Case Design from the Independent Paths

Test case ID	Input Number	Expected Result	Independent path covered by the test case.
1	1	No output is displayed	1-2-3-4-5-10-12
2	2	Prime number	1-2-3-4-5-10-11-12
3	4	Not a prime number	1-2-3-4-5-6-7-8-10-12
4	3	Prime Number	1-2-3-4-5-6-9-5-10-11-12