# Software Testing and Quality Assurance

## Module-3

Control Flow Testing

By Prof. Pallavi Mahajan

# Contains

- Control Flow Testing:
- Outline of Control Flow Testing
- Control Flow Graph, Paths in a Control Flow Graph
- Path Selection Criteria
- All-Path Coverage Criterion
- Statement Coverage Criterion
- Branch Coverage Criterion
- Predicate Coverage Criterion
- Generating Test Input, Examples of Test Data Selection

# Control Flow Testing

# Introduction to Control Flow Testing

➢ **Control Flow in Programs**

- Control flow refers to the order in which program instructions are executed.
- Conditional statements (e.g., if(), for(), while(), goto()) alter the default sequential control flow.

➢ **Purpose of Control Flow Testing:**

- To ensure that the program follows the expected paths and produces the desired outcomes.

# Introduction to Control Flow Testing

## Basic Program Statements

- **Types of Basic Statements:**
  - **Assignment Statements**:
    - e.g., `x = 2 * y;`
    - Used to assign values to variables.
  - **Conditional Statements**:
    - e.g., `if(x != y)`
    - Used for decision-making, impacting program flow.

# Introduction to Control Flow Testing

## Program Flow and Conditional Statements

- **Sequential Control Flow:**

  *Without conditional statements, program instructions execute in sequence.*

- **Altered Control Flow with Conditionals:**

  *Conditional statements introduce complexity by modifying the flow, making program execution non-linear.*

# Introduction to Control Flow Testing

## Function Calls and Program Flow

### Function Calls and Control Flow:

- Function calls provide abstraction and lead to control entering the function.
- When the function returns, control exits.

### Program Path:

- Defined as the execution of instructions from entry to exit in a program unit (e.g., a function).
- Each program path is characterized by specific input and expected output.

# Introduction to Control Flow Testing

## Path Coverage in Control Flow Testing

### Large Number of Paths:

- **A program can have many potential execution paths, depending on the input values.**

### Goal of Control Flow Testing:

- **Test as many paths as necessary, but with optimal coverage to identify defects efficiently.**
- **Cost vs. Effectiveness:** Executing many paths can be costly and not always effective.

# Introduction to Control Flow Testing

## Key Concepts in Control Flow Testing

### Program Path:

- A sequence of statements in the program.
- A specific execution of the program is characterized by an input leading to a particular path.

### Control Flow Testing:

- A structural testing method used by developers to test the code they've written.
- The focus is on testing small units of code (e.g., a function) rather than the entire program.

# Introduction to Control Flow Testing

## Tools for Control Flow Testing

### Automated Tools:

- **Tools identify paths based on user-defined criteria.**
- **Generate corresponding inputs to execute paths.**
- **Can also generate stubs and drivers to facilitate testing.**

### Benefits:

•Tools improve testing efficiency by automating path selection and test execution.

# Introduction to Control Flow Testing

## Structural vs. Semantic Testing

### Structural Testing:

- Testing based on the program's internal structure (e.g., paths within a function).

### Semantic Testing:

- Testing based on the program's behavior during execution.
- For each input, the program unit may execute a different path.

# Introduction to Control Flow Testing

**Key Points:**

- **Control flow testing focuses on selecting paths in a program and ensuring expected outcomes.**
- **It is a crucial method for structural testing of code, particularly small units like functions.**
- **Effective control flow testing minimizes cost while maximizing defect identification.**

# Outline of Control Flow Testing

# Outline of Control Flow Testing

**Control Flow Testing Overview**

- A type of structural testing to assess program paths and their execution.
- The goal is to select paths from a program unit to test various conditions and ensure expected behavior.

# Outline of Control Flow Testing

## Process of Generating Test Input Data

### Overview (as depicted in Figure 4.1):

- The process involves several activities to generate test input data based on path selection criteria.

### Inputs Required:

- Source Code of a Program Unit
- Path Selection Criteria: Criteria determine which paths to test.

# Outline of Control Flow Testing

## Path Selection Criteria Examples

### Example 1: Every Statement Executed at Least Once

- **Select paths ensuring that every statement in the program is executed at least once.**

### Example 2: Conditional Statements Evaluated to True and False

- **Select paths where conditional statements (e.g., if() statements) evaluate to both true and false at least once.**

# Outline of Control Flow Testing

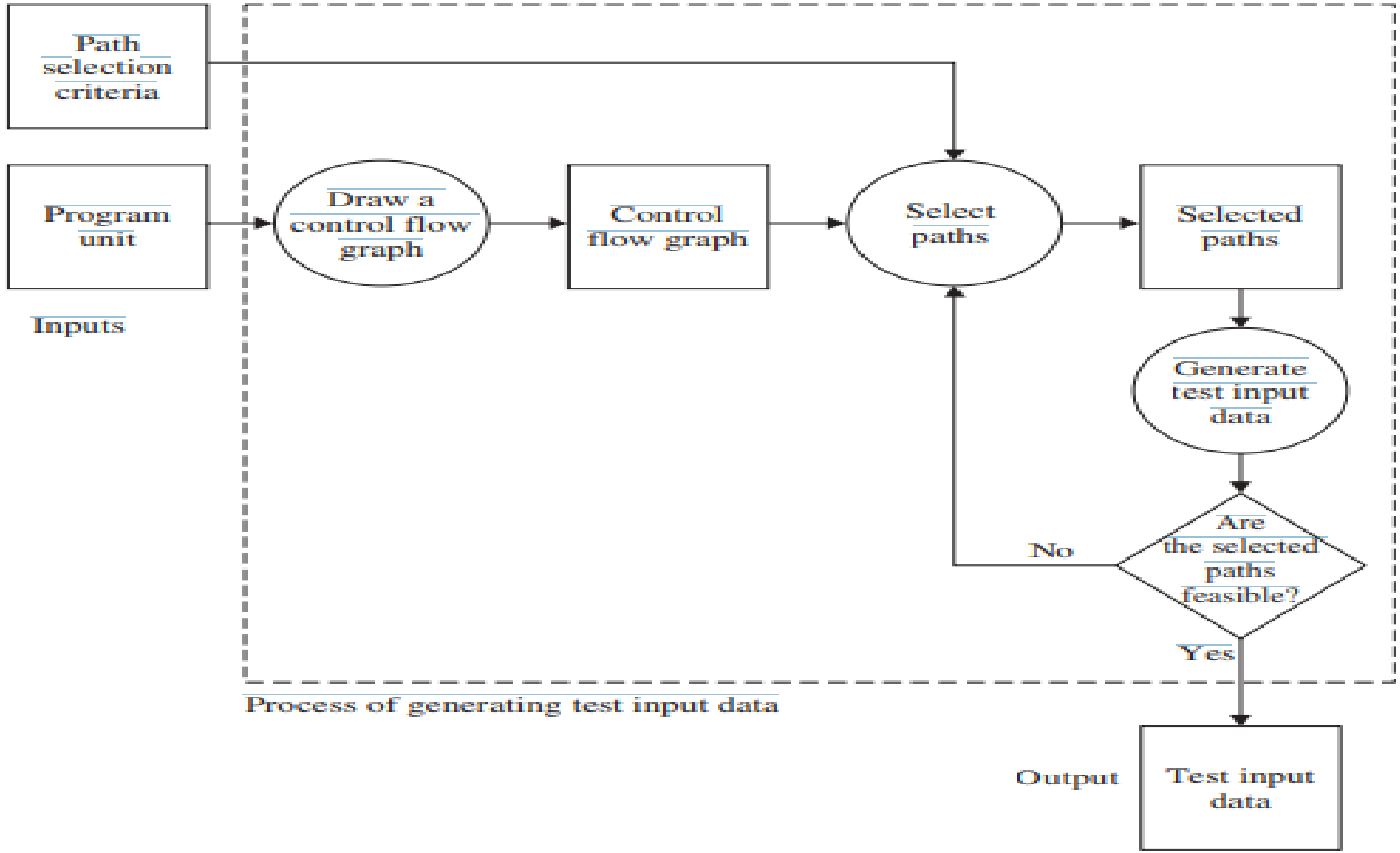


Figure 4.1 Process of generating test input data for control flow testing.

# Outline of Control Flow Testing

## Generation of a Control Flow Graph (CFG)

- **Control Flow Graph (CFG)**:
  - A detailed graphical representation of the program unit.
  - Used to visualize all possible paths.
- **Automated Test Generation**:
  - A compiler can be modified to automatically generate a CFG for the program unit.

# Outline of Control Flow Testing

## Selection of Paths

**Path Selection from CFG**:

- Paths are selected to meet the path selection criteria.
- The structure of the CFG is considered when choosing paths to test.

# Outline of Control Flow Testing

## Generation of Test Input Data

- **Test Input Data Generation**:
  - Paths can only be executed if specific input values satisfy all conditional statements along the path.
  - Each path has a set of input values that make it feasible for execution.
- **Feasible Path**:
  - A feasible path is one where the chosen input values allow all conditions to evaluate as needed.
- **Infeasible Path**:
  - A path is infeasible if the selected input values cannot satisfy the required conditions.

# Outline of Control Flow Testing

## Feasibility Test of a Path

- **Checking Path Feasibility**:
  - A key part of control flow testing is checking whether selected paths are feasible.
- **If a Path is Infeasible**:
  - If a selected path is found to be infeasible, alternative paths are selected to meet the selection criteria.

# Control Flow Graph, Paths in a Control Flow Graph

# Control Flow Graph

## Introduction to CFG

- **Definition: A CFG is a graphical representation of a program unit.**
- **Purpose: Helps in understanding the flow of execution in a program.**
- **Symbols Used: Sequential computation, decision point, and merge point.**

# Control Flow Graph, Paths in a Control Flow Graph

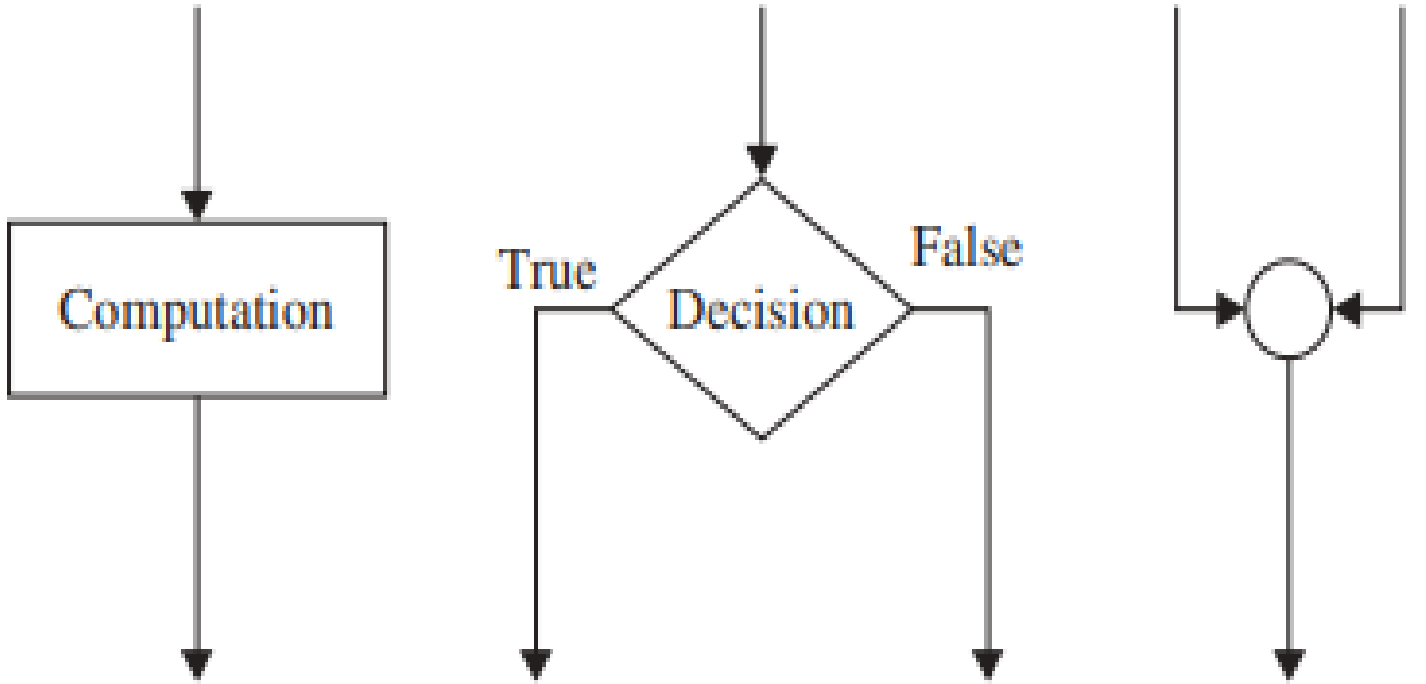


Figure 4.2 Symbols in a CFG.

# Control Flow Graph, Paths in a Control Flow Graph

## Symbols in CFG

- Rectangle: Represents a sequential computation.
- Diamond: Represents a decision point (True/False paths).
- Merge Node: Not explicitly labeled but inferred from paths.
- Diagram: Illustration of CFG symbols.

# Control Flow Graph

## Example CFG - openfiles() Function

```
FILE *fptr1, *fptr2, *fptr3; /* These are global variables. */

int openfiles(){
    /*
       This function tries to open files "file1", "file2", and
       "file3" for read access, and returns the number of files
       successfully opened. The file pointers of the opened files
       are put in the global variables.
    */
      int i = 0;
      if(
          ((( fptr1 = fopen("file1", "r")) != NULL) && (i++)
                                                    && (0)) ||
          ((( fptr2 = fopen("file2", "r")) != NULL) && (i++)
                                                    && (0)) ||
          ((( fptr3 = fopen("file3", "r")) != NULL) && (i++))
      );
      return(i);
}
```

Figure 4.3 Function to open three files.

# Control Flow Graph, Paths in a Control Flow Graph

## Example CFG - openfiles() Function

- **Description: Function to open three files and return the count of successful opens.**
- **Code Snippet: Display the openfiles() function.**
- **Explanation: Conditional checks within the if statement.**

# Control Flow Graph, Paths in a Control Flow Graph

## High-Level CFG of openfiles()

- **CFG Diagram: Simple CFG with three numbered nodes (1, 2, 3).**
- **Explanation:**
- **Node 1: Initialization (i = 0)**
- **Node 2: Conditional check**
- **Node 3: Return statement**

# Control Flow Graph, Paths in a Control Flow Graph

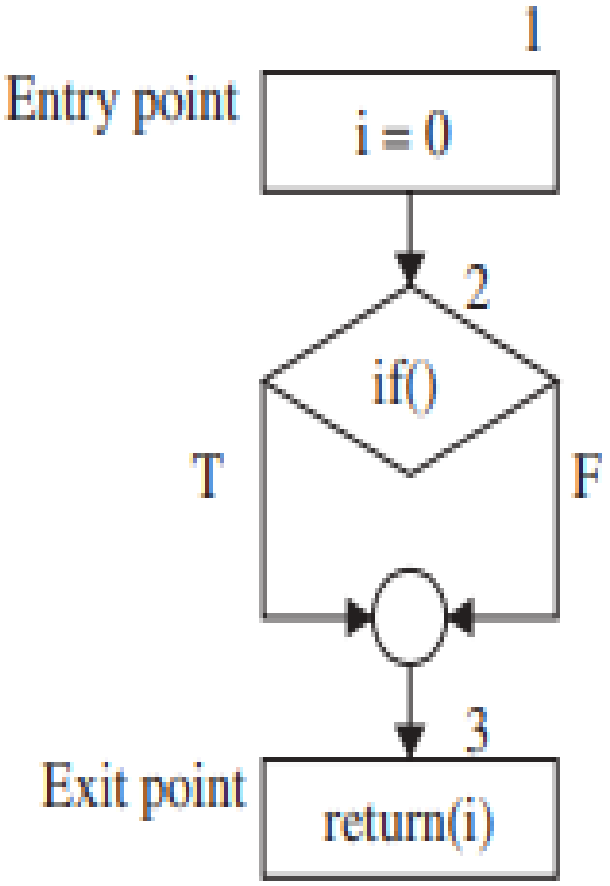


Figure 4.4 High-level CFG representation of openfiles(). The three nodes are numbered 1, 2, and 3.

# Control Flow Graph, Paths in a Control Flow Graph

## Detailed CFG of openfiles()

- **Complex CFG Representation:**
- **Shows multiple decision paths**
- **21 numbered nodes with multiple branches**
- **Diagram: Detailed CFG for openfiles()**

# Control Flow Graph, Paths in a Control Flow Graph

## Execution of Conditional Statements in openfiles()

- **Breakdown of execution order:**
- **Assignment statements: fptr1 = fopen(“file1”, “r”), i++**
- **Boolean and relational operators: &&, ||, fptr1 != NULL**
- **Short-circuit evaluation explained.**
- **Example: Step-by-step execution.**

# Control Flow Graph, Paths in a Control Flow Graph

## ReturnAverage() Function Example

- **Java Function: Computes average of numbers within [MIN, MAX] range.**
- **Code Snippet: Display the ReturnAverage() function.**
- **Explanation:**
- **Iterates through an array to compute the average.**
- **Handles array size constraints using -999.**

# Control Flow Graph, Paths in a Control Flow Graph

## CFG Representation of ReturnAverage()

- **CFG Diagram: Shows control flow for ReturnAverage().**
- **Explanation:**
- **Entry node initializes variables.**
- **Loop processes array values.**
- **Decision nodes handle conditions and computations.**

# Control Flow Graph, Paths in a Control Flow Graph

## Paths in a CFG

- **Definition: A path is a sequence of computation and decision nodes from entry to exit.**
- **Entry and Exit Nodes: Each CFG has one entry and one exit.**
- **Example Paths: Show different paths taken in ReturnAverage().**
- **Table: Sample paths with unfolded loops.**

# Control Flow Graph, Paths in a Control Flow Graph

## Key Points

- ***Summary:***
  - CFG helps in analyzing program flow and testing paths.
  - Identifies key decision points and execution paths.
- ***Applications:***
  - Used in software testing and debugging.
  - Basis for path-based test coverage techniques.

# Path Selection Criteria

## Path Selection Criteria

- Selecting paths based on statement, branch, and predicate coverage.

# Path Selection Criteria

## Introduction to Path Selection Criteria

- **Definition: A CFG can have numerous execution paths.**
- **Purpose: Selecting meaningful paths for effective testing.**
- **Importance: Helps in identifying defects and optimizing test cases.**

# Path Selection Criteria

## CFG Representation of ReturnAverage()

- **Diagram: Display Figure 4.7 (CFG of ReturnAverage()).**
- **Explanation:**
- **Entry node initializes variables.**
- **Loop processes array values.**
- **Decision nodes evaluate conditions.**

# Path Selection Criteria

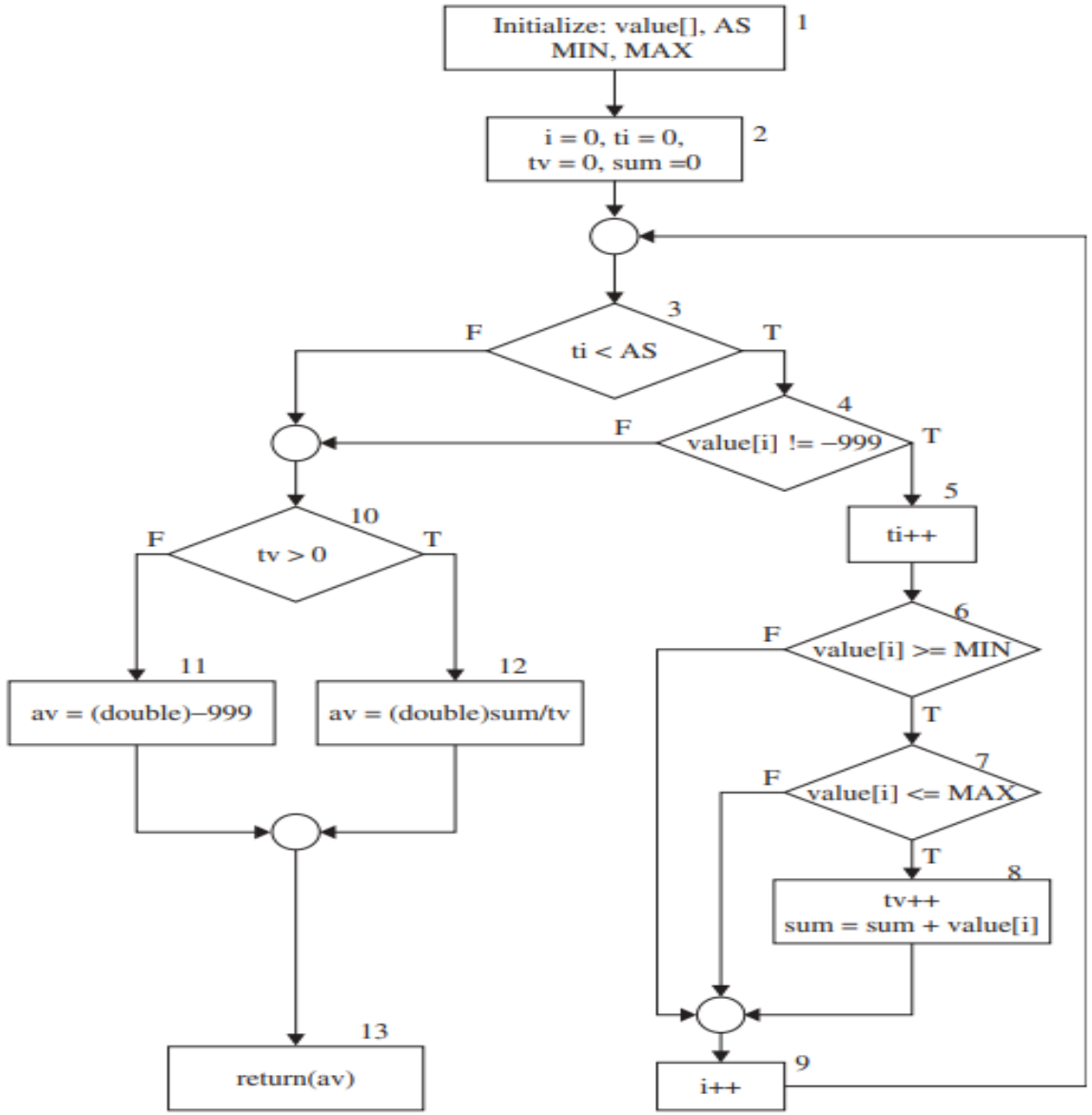


Figure 4.7 A CFG representation of ReturnAverage(). Numbers 1–13 are the nodes.

# Path Selection Criteria

**Example Paths in CFG**

- **Table: Examples of paths (1-13) from Figure 4.7.**
- **Explanation:**
- **Different paths traverse through various program conditions.**
- **Loops and conditions impact path selection.**

# Path Selection Criteria

TABLE 4.1 Examples of Path in CFG of Figure 4.7

| | |
|---|---|
| Path 1 | 1-2-3(F)-10(T)-12-13 |
| Path 2 | 1-2-3(F)-10(F)-11-13 |
| Path 3 | 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13 |
| Path 4 | 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13 |

# Path Selection Criteria

## Importance of Path Selection

**Why Select Paths?**

- Testing every path may be impractical for complex programs.
- Optimized path selection improves efficiency.

**Goals:**

- Ensure program constructs are exercised at least once.
- Avoid redundant path execution.
- Track tested vs. untested features.

# Path Selection Criteria

## Well-Known Path Selection Criteria

- **Select all paths.**
- **Select paths for statement coverage.**
- **Select paths for branch coverage.**
- **Select paths for predicate coverage.**

# Path Selection Criteria

## 1. All-Path Coverage Criterion

**1.Definition: Select all feasible paths.**
**2.Example: openfiles() function with multiple paths.**
**3.Challenge: Not all paths may be feasible in large programs.**

# Path Selection Criteria

## 2. Statement Coverage Criterion

- **Definition: Execute every statement at least once.**
- **Example:**
- **Nodes 2, 3, 4 in ReturnAverage() must be covered.**
- **Loop control conditions at nodes 3 and 4 must be evaluated.**
- **Selection Strategy:**
- **Choose minimal paths that cover all nodes.**

# Path Selection Criteria

## 3. Branch Coverage Criterion

- **Definition: Ensure every decision outcome is tested.**
- **Example: True/False branches at nodes 3, 4, and 10 in ReturnAverage().**
- **Uncovered Branches:**
- **Highlighted in CFG of Figure 4.8.**
- **Requires additional test cases for full coverage.**

# Path Selection Criteria

## 4. Predicate Coverage Criterion

- **Definition: Ensure all Boolean conditions are tested in all scenarios.**
- **Example:**
- **Testing OR condition (OB1 || OB2 || OB3) in Figure 4.9a.**
- **Testing AND condition (AB1 && AB2 && AB3) in Figure 4.9b.**
- **Requires additional test cases to capture all evaluations.**

# Key Points- Path Selection Criteria

- **Statement Coverage: Covers all statements but may miss some branches.**
- **Branch Coverage: Ensures all decisions are evaluated.**
- **Predicate Coverage: Tests all possible logical outcomes.**
- **Trade-offs: Higher coverage requires more test cases.**

# Key Points- Path Selection Criteria

- **CFG-based path selection improves software testing efficiency.**
- **A balance between complete coverage and practical feasibility is needed.**
- **Used in debugging, test case generation, and program verification.**

# Generating Test Input, Examples of Test Data Selection

# Generating Test Input

## Introduction to Test Input Generation

- **Definition: Identifying input values to execute selected paths in a CFG.**
- **Purpose: Ensuring that test cases trigger specific execution paths.**
- **Key Terms: Input vectors, predicates, path predicates.**

# Generating Test Input

## Understanding Input Vectors

- Definition: A collection of all data entities read by a routine before execution.
- Types of Input Vectors:
- Input arguments to a routine.
- Global variables and constants.
- Files, registers, network connections, timers.
- Example: Input vector for openfiles() function.

# Generating Test Input

## Understanding Predicates in CFG

- **Definition: A logical function evaluated at a decision point.**
- **Examples:**
- **ti < AS (Decision node 3 in Figure 4.7)**
- **OB (Decision node 5 in Figure 4.9)**

# Generating Test Input

## Path Predicates in CFG

- **Definition: A set of predicates associated with a specific path.**
- **Example: Figure 4.10 showing decision nodes along a path.**
- **Importance: Helps in controlling execution flow in test cases.**

# Generating Test Input

## Evaluating Path Predicate Expressions

- Definition: A rewritten form of path predicates using input vector elements.
- Example: Figure 4.11 showing a path predicate for a selected path.
- Importance: Differentiates between multiple instances of the same predicate.

# Generating Test Input

## Predicate Interpretation and Symbolic Substitution

- **Rewriting predicates using input vector elements.**
- **Example:**
- **Original predicate: x1 + y >= 0**
- **Substituted form: x1 + x2 + 7 >= 0**
- **Importance: Eliminates dependence on local variables.**

# Generating Test Input

## Path Predicate Expression Properties

- Only contains elements from input vectors and constants.
- Expresses constraints that must be met for path execution.
- Used to generate test inputs by solving constraint equations.
- Identifies infeasible paths if constraints cannot be satisfied.

# Generating Test Input

## Example of Path Predicate Interpretation

- **Table showing interpreted predicates for a selected path (Figure 4.10).**
- **Understanding evaluations like value[i] != -999 ≡ True.**
- **Ensuring accurate test execution by mapping conditions.**

# Generating Test Input

## Handling Infeasible Paths

- **Example: Figure 4.14 showing an infeasible path.**
- **Constraint example: 0 > 0 ≡ True (unsatisfiable condition).**
- **Conclusion: If a path predicate expression cannot be satisfied, the path is infeasible.**

# Generating Test Input

## Generating Test Data for Path Execution

- **Step 1: Solve constraints to determine valid input values.**
- **Step 2: Identify necessary values for MIN, MAX, and value[i].**
- **Step 3: Verify path execution with generated inputs.**
- **Example: Figure 4.16 showing a valid test data set.**

# Generating Test Input

## Key Points of Test Input Generation Strategies

- **Input Vectors: Identify relevant input sources.**
- **Predicates: Understand conditions affecting execution.**
- **Path Expressions: Solve constraints to generate inputs.**
- **Handling Infeasibility: Identify alternative paths when needed.**

# Generating Test Input

## Key Points of Test Input Generation Strategies

- **Effective test input generation ensures proper CFG coverage.**
- **Helps in debugging, verification, and test case design.**
- **Crucial for software reliability and defect detection.**

# Generating Test Input

- Typ

# Thank You..