CUI Abbottabad

Department of Computer Science

SOFTWARE TESTING

Lecture 9

Whitebox Testing Technique
Data Flow Testing

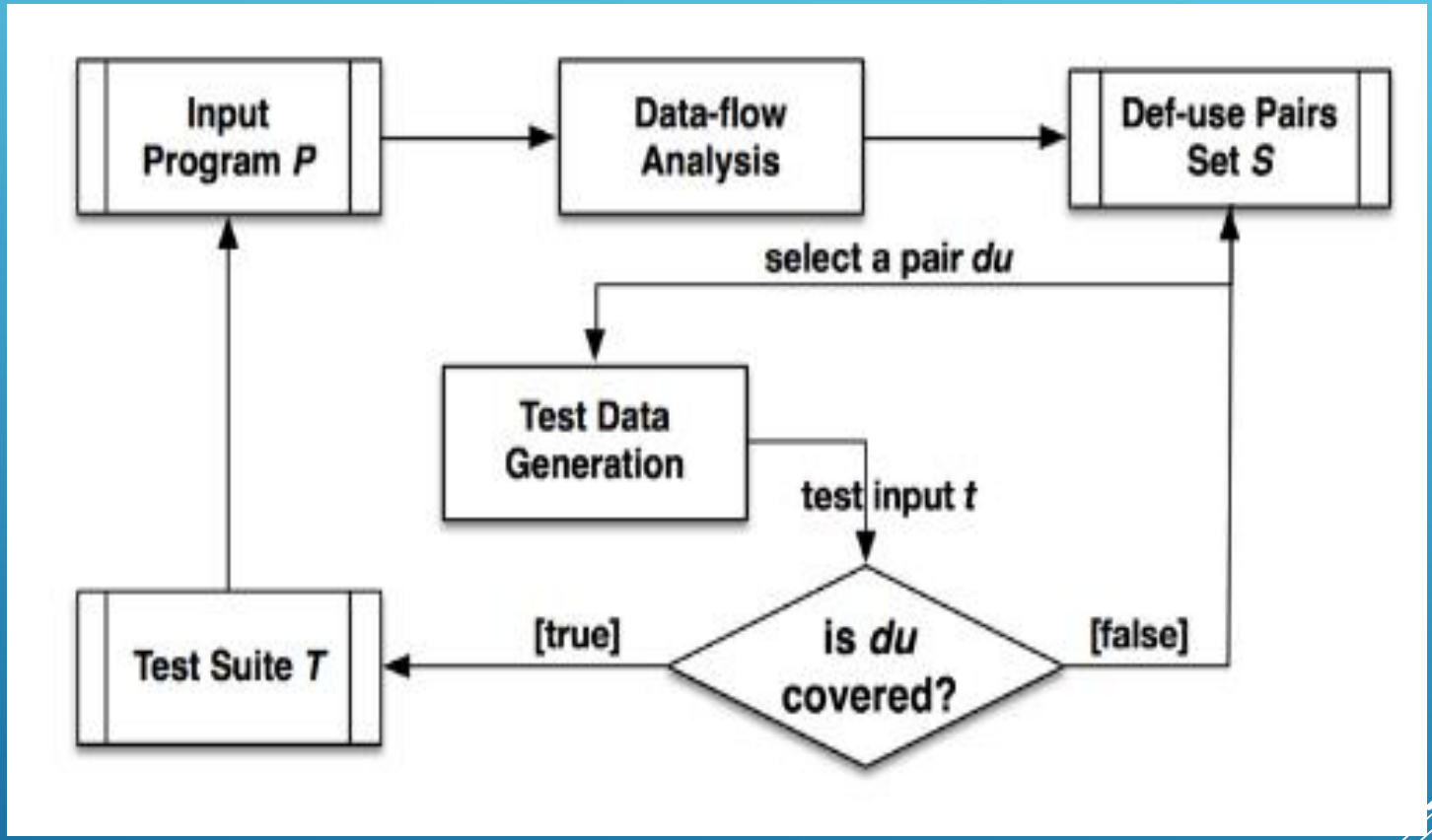
DATA FLOW TESTING

- A program unit, such as a function, accepts input values, performs computations, assign new values to variables, and produces output values.
- Idea: A programmer can perform a number of tests on data values.
  - These tests are collectively known as data flow testing.
- One can visualize the “flow” of data values from one statement to another.
- A data value produced in one statement is expected to be used later.
  - Example: Obtain a file pointer ……. use it later. If later use is never verified, we do not know if earlier assignment is acceptable.

DATA FLOW TESTING

- Data flow testing uses the control flow graph to explore the unreasonable things that can happen to data (data flow anomalies).
- T**wo motivations** of data flow testing:
  - The memory location for a variable is accessed in a “desirable” way.
  - Verify the correctness of data values “defined” or generated for a variable and observe all the “uses” of the value produce the desired results.

DATA FLOW TESTING



STEPS OF DATA FLOW TESTING

- Draw a data flow graph from a program.
- Select one or more data flow testing criteria.
- Identify paths in the data flow graph satisfying the selection criteria.
- Derive path predicate expressions from the selected paths and solve those expressions to derive test input.

TWO CONCEPTUAL LEVELS

Data flow testing can be performed at two conceptual levels.

- **Static data flow testing**
  - Identify potential defects, commonly known as data flow anomaly.
  - Testing is performed by analyzing the source code, and it does not involve actual execution of source code..
- **Dynamic data flow testing**
  - Involves actual program execution.
  - Bears similarity with control flow testing.
    - Identify paths to execute them.
    - Paths are identified based on data flow testing criteria.

DATA FLOW ANOMALY

- **Anomaly:** It is an abnormal way of doing something.
  - Example 1: The second definition of x overrides the first.
  - x = f1(y);
  - x = f2(z);
- Data flow anomalies are detected based on the associations between values and variables.
  - Variables are used without being initialized.
  - Initialized variables are not used once.

DATA FLOW ANOMALIES

Three types of abnormal situations with using variable.

- Defined and Then Defined Again
- Undefined but Referenced
- Defined but Not Referenced

- Undefined: Declare
- Defined: Initialize
- Referenced: declare, initialize and utilize

1. DEFINED AND THEN DEFINED AGAIN

- The computation performed by the first statement is redundant if the second statement performs the intended computation.
- Example 1:

i. x = f1(y);

ii. x = f2(z);

- Four interpretations of Example 1
  - The first statement is redundant.
  - The first statement has a fault -- the intended one might be: w = f1(y).
  - The second statement has a fault – the intended one might be: v = f2(z).
  - There is a missing statement in between the two: v = f3(x).
- Note: It is for the programmer to make the desired interpretation.

2. UNDEFINED BUT REFERENCED

- To use an undefined variable in a computation.
- Example: $x = x - y - w;$
  - where the variable *w* has not been *initialized/defined* by the programmer.
- Two interpretations:
  - The programmer made a mistake in using w.
  - The programmer wants to use the compiler assigned value of w.
- For correction: one must eliminate the anomaly either by initializing w or replacing w with the intended variable.

3. DEFINED BUT NOT REFERENCED

- Define a variable and then to undefined it without using it in any later computation.
- For example, $x = f(x, y)$
  - Here new value is assigned to the variable x. If the value of x is not used in any later computation, then we should be suspicious of the computation represented by $x = f(x, y)$.

DATA FLOW GRAPH (DIRECTED GRAPH)

- A sequence of *definitions* and *c-uses* is associated with each node of the graph.
- A set of *p-uses* is associated with each *edge* of the graph.
- The *entry node* has a definition of each parameter and each nonlocal variable which occurs in the subprogram.
- The *exit node* has an *undefinition* of each local variable.

TWO FORMS OF *USES* OF A VARIABLE

- **Computation use (c-use):** In c-use, a potentially new value of another variable or of the same variable is produced.
  - Example: x = 2*y; /* y has been used to compute a value of x. */
- **Predicate use (p-use):** Here the use of a variable in a predicate controlling the flow of execution.
  - Example: if (y > 100) { …} /* y has been used in a condition. */

EXAMPLE

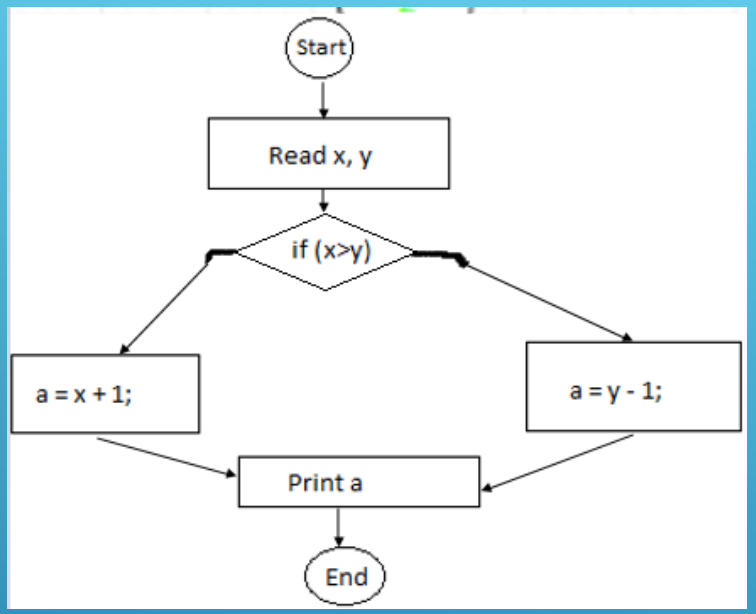
```
1. read x, y;
2. if(x>y)
3. a = x+1
else
4. a = y-1
5. print a;
```

EXAMPLE DFG

1. read x, y;
2. if(x>y)
3. a = x+1

else

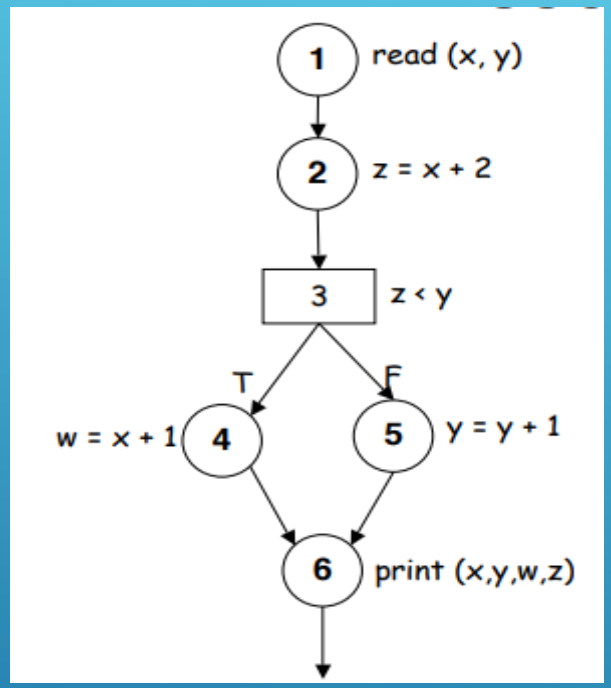
4. a = y-1
5. print a;



Variable	Variable Defined at node	Variable Used at node
x	1	2, 3
y	1	2, 4
a	3, 4	5

ANOTHER EXAMPLE DFG AND P & C USE OF VARIABLE

```
1.  read (x, y);
2.  z = x + 2;
3.  if (z < y)
4.        w = x + 1;
    else
5.        y = y + 1;
6.  print (x, y, w, z);
```



Def	C-use	P-use
x, y		
z	x	
		z, y
w	x	
y	y	
	x, y, w, z	

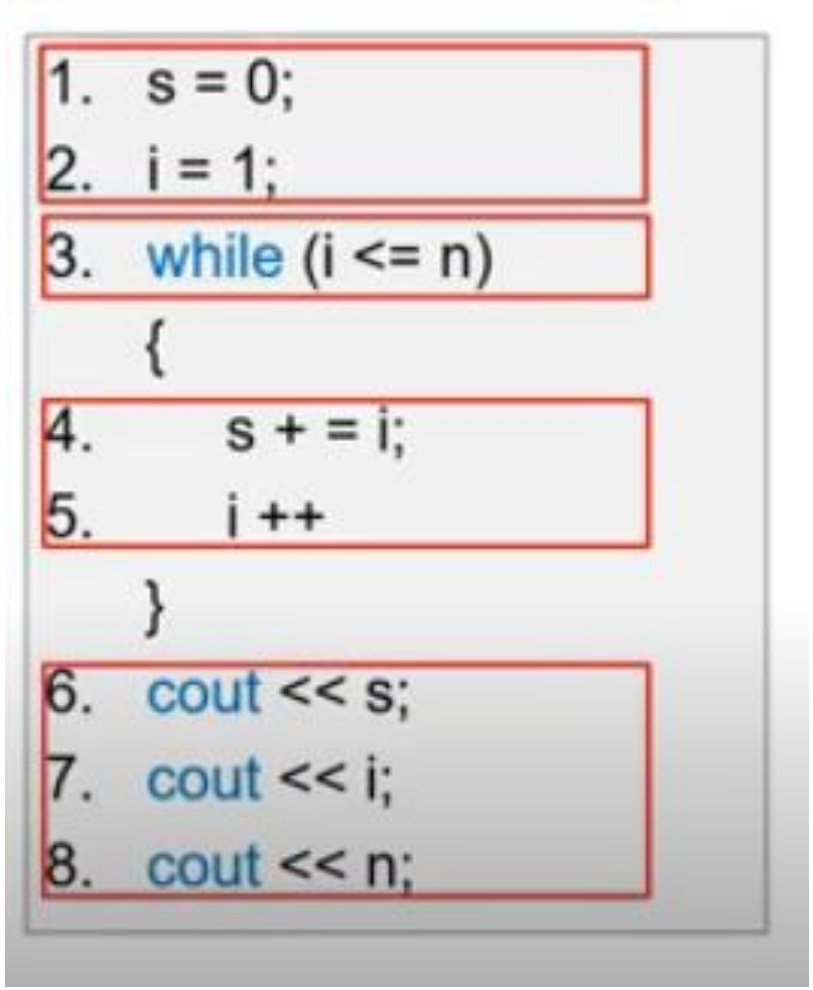
EXAMPLE

- Identify the basic blocks
- Identify all the definitions
  - Where variables get their values
- Identify all the uses
  - Where variables are used
  - Indicate p-uses (in predicates)
  - Indicate c-uses (in computations)
- Draw path from each definition to each use that might get data from that definition

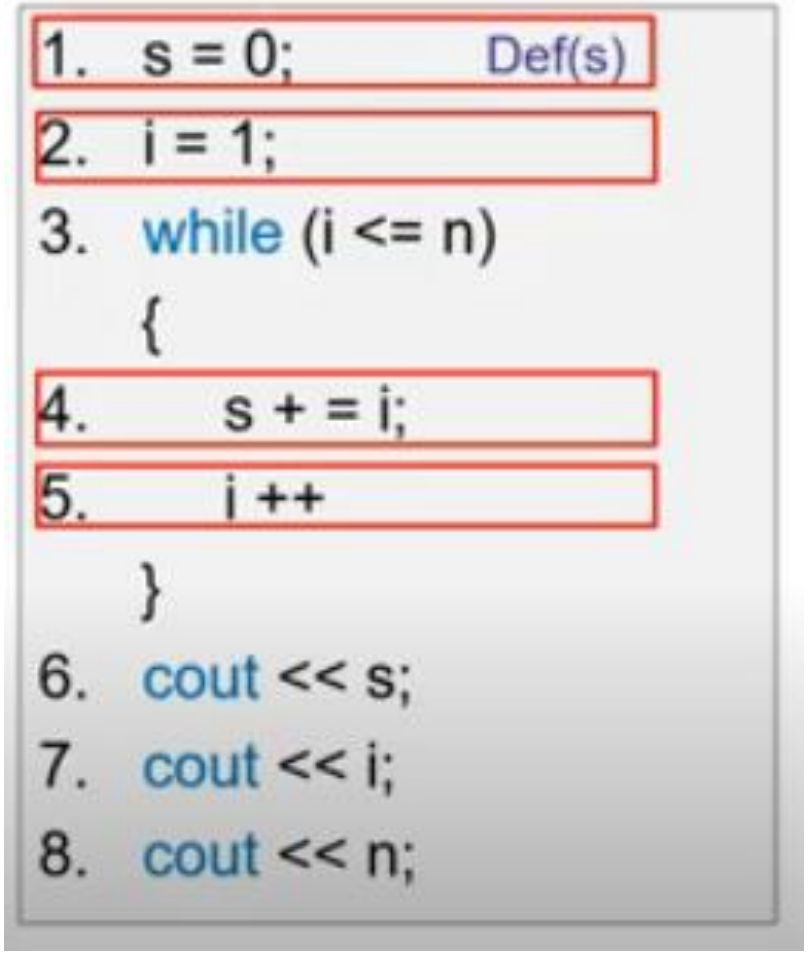
EXAMPLE CODE

```
1.  s = 0;
2.  i = 1;
3.  while (i <= n)
    {
4.      s + = i;
5.      i ++
    }
6.  cout << s;
7.  cout << i;
8.  cout << n;
```

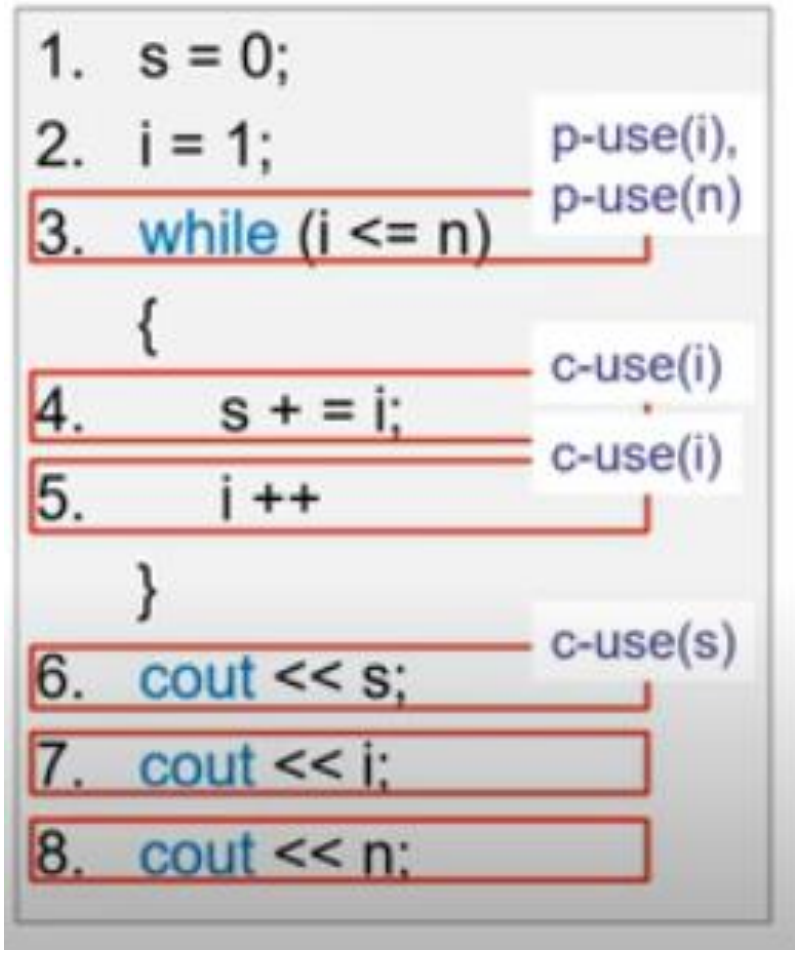
IDENTIFY BLOCKS



IDENTIFY DEFINITIONS



IDENTIFY USES (C-USE & P-USE)



DATA FLOW TESTING CRITERIA

- All-defs
- All-c-uses
- All-p-uses
- All-p-uses/some-c-uses
- All-c-uses/some-p-uses
- All-uses
- All-du-paths

CFT AND DFT

There is much similarity between control flow testing and data flow testing. Moreover, there is a key difference between the two approaches.

- The similarities stem from the fact that both approaches identify program paths and emphasize on generating test cases from those program paths.
- The difference between the two lies in the fact that control flow test selection criteria are used in the former, whereas data flow test selection criteria are used in the latter approach.

REFERENCES

Book:

Chapter 5: Software Testing and Quality Assurance (Indian)