SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

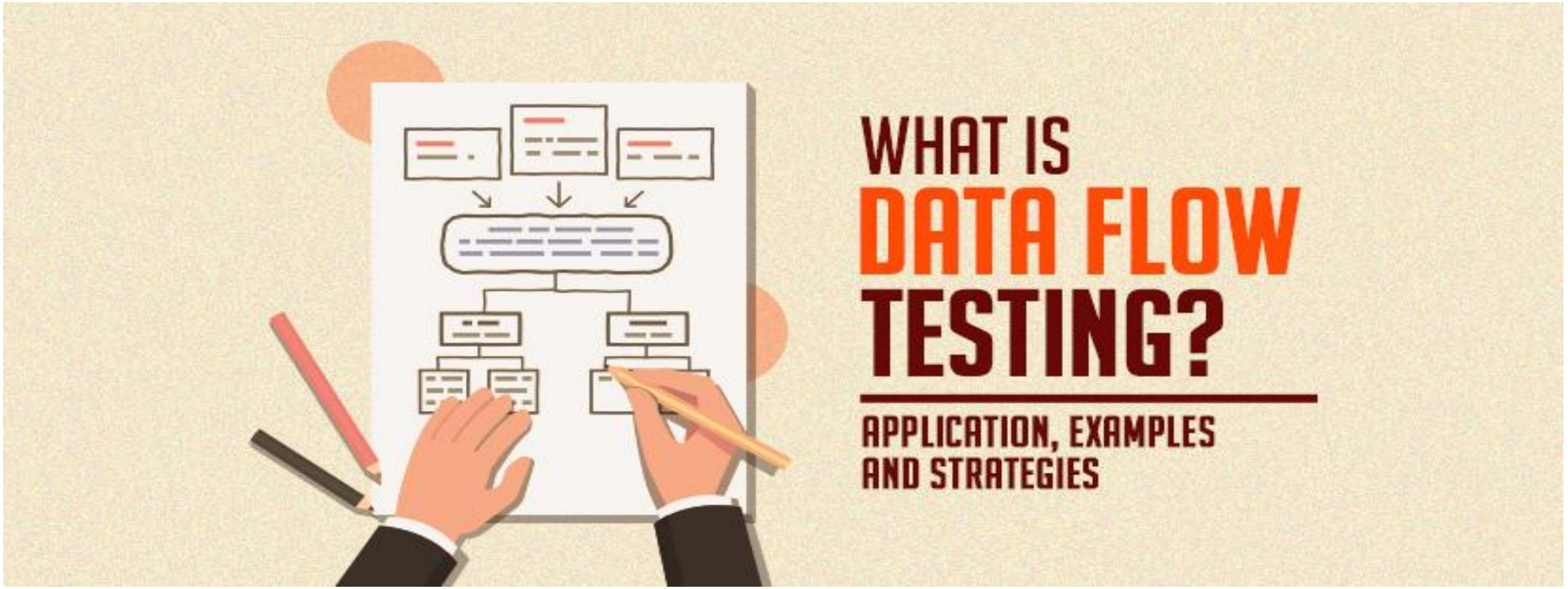
Part II-Software Testing

Structural Testing

Lecture # 25, 26, 27
26,27,28 Oct

TODAY'S OUTLINE

- Structural Testing
  - Data Flow Testing
  - Slice based testing
  - Quiz


WHAT IS
DATA FLOW
TESTING?
APPLICATION, EXAMPLES
AND STRATEGIES

DATA FLOW TESTING

- In control flow testing, we find various paths of a program and design test cases to execute those paths.
- We may like to execute every statement of the program at least once before the completion of testing.
- Consider the following program:

```
1. # include < stdio.h>
2. void main ()
3. {
4. int a, b, c;
5. a = b + c;
6. printf ("%d", a);
7. }
```

DATA FLOW TESTING

- Data flow testing may help us to minimize such mistakes. It is done to cover the path testing and branch testing gap.
- It has nothing to do with dataflow diagrams. It is based on variables, their usage and their definition(s) (assignment) in the program.
- The main points of concern are:
  - Statements where these values are used (referenced).
  - Statements where variables receive values (definition).
- Data flow testing focuses on variable definition and variable usage.
- The process is conducted to detect the bugs because of the incorrect usage of data variables or data values.

DEFINE/REFERENCE ANOMALIES

- Some of the define / reference anomalies are given as:
  - A variable is defined but never used / referenced.
  - A variable is used but never defined.
  - A variable is defined twice before it is used.
  - A variable is used before even first-definition.
- Define / reference anomalies may be identified by static analysis of the program.

DATA FLOW TESTING TERMS DEFINITIONS

- A program is first converted into a program graph.
- Defining node
- A node of a program graph is a defining node for a variable , if and only if, the value of the variable is defined in the statement corresponding to that node. It is represented as DEF (, n) where is the variable and n is the node corresponding to the statement in which is defined.
- Usage node
- A node of a program graph is a usage node for a variable , if and only if, the value of the variable is used in the statement corresponding to that node. It is represented as USE (, n), where ‘ ’ is the variable and ‘n’ in the node corresponding to the statement in which ‘ ’ is used.
- A usage node USE (, n) is a predicate use node (denoted as P-use), if and only if, the statement corresponding to node ‘n’ is a predicate statement otherwise USE (, n) is a computation use node (denoted as C-use).

DATA FLOW TESTING TERMS DEFINITIONS

- Definition use Path
- A definition use path (denoted as du-path) for a variable ‘ ’ is a path between two nodes ‘m’ and ‘n’ where ‘m’ is the initial node in the path but the defining node for variable ‘ ’ (denoted as DEF (, m)) and ‘n’ is the final node in the path but usage node for variable ‘ ’ (denoted as USE (, n)).

IDENTIFICATION OF DU PATHS

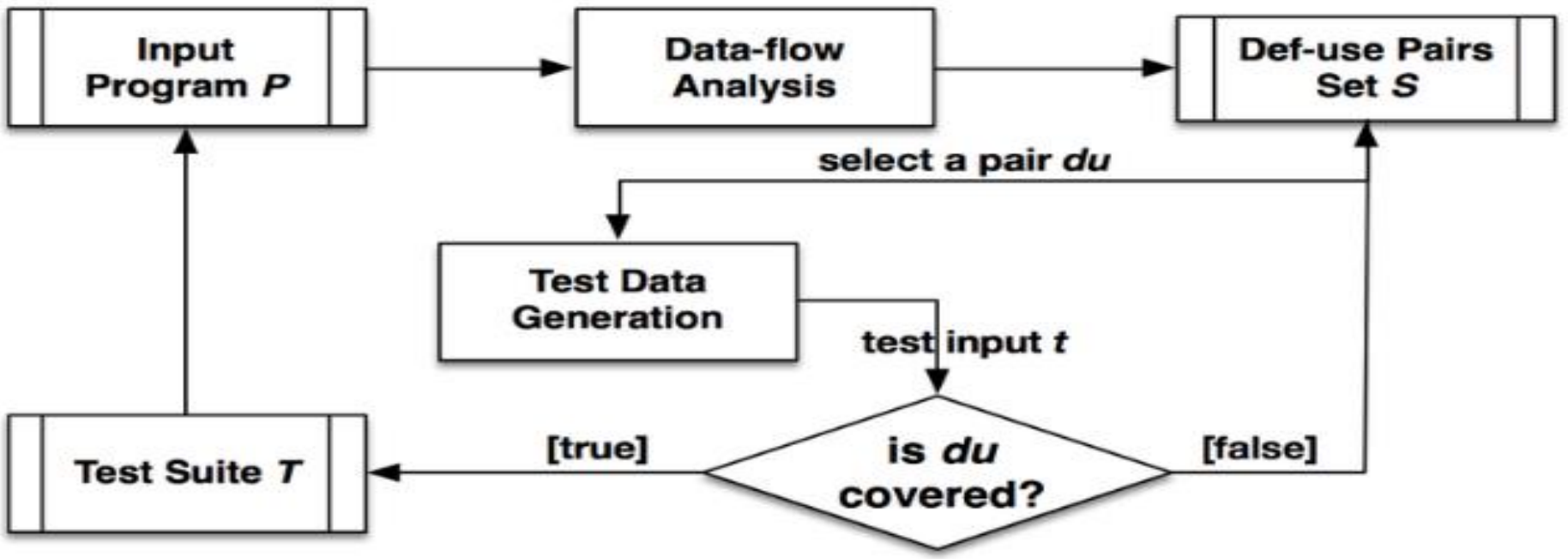
- The various steps for the identification of du and dc paths are given as:
- Draw the program graph of the program.
- Find all variables of the program and prepare a table for define / use status of all variables using the following format:

S. No.	Variable(s)	Defined at node	Used at node

- Generate all du-paths from define/use variable table of above step using the following

S. No.	Variable	du-path(begin, end)

DATA FLOW TESTING CYCLE



TESTING STRATEGIES USING DU-PATHS

- We want to generate test cases which trace every definition to each of its use and every use is traced to each of its definition. Some of the testing strategies are given as:
- **Test all du-paths**
- All du-paths generated for all variables are tested. This is the strongest data flow testing strategy covering all possible du-paths.
- **Test all uses**
- Find at least one path from every definition of every variable to every use of that variable which can be reached by that definition.
- For every use of a variable, there is a path from the definition of that variable to the use of that variable.

TESTING STRATEGIES USING DU-PATHS

- **Test all definitions**
- Find paths from every definition of every variable to at least one use of that variable;
- The first requires that each definition reaches all possible uses through all possible du-paths, the second requires that each definition reaches all possible uses, and the third requires that each definition reaches at least one use.

TYPES OF DATA FLOW TESTING

- Static Data Flow Testing
- No actual execution of the code is carried out in Static Data Flow testing. Generally, the definition, and usage pattern of the data variables is scrutinized through a control flow graph.
- Dynamic Data Flow Testing
- The code is executed to observe the transitional results. Dynamic data flow testing includes:
  - Identification of definition and usage of data variables.
  - Identifying viable paths between definition and usage pairs of data variables.
  - Designing & crafting test cases for these paths.

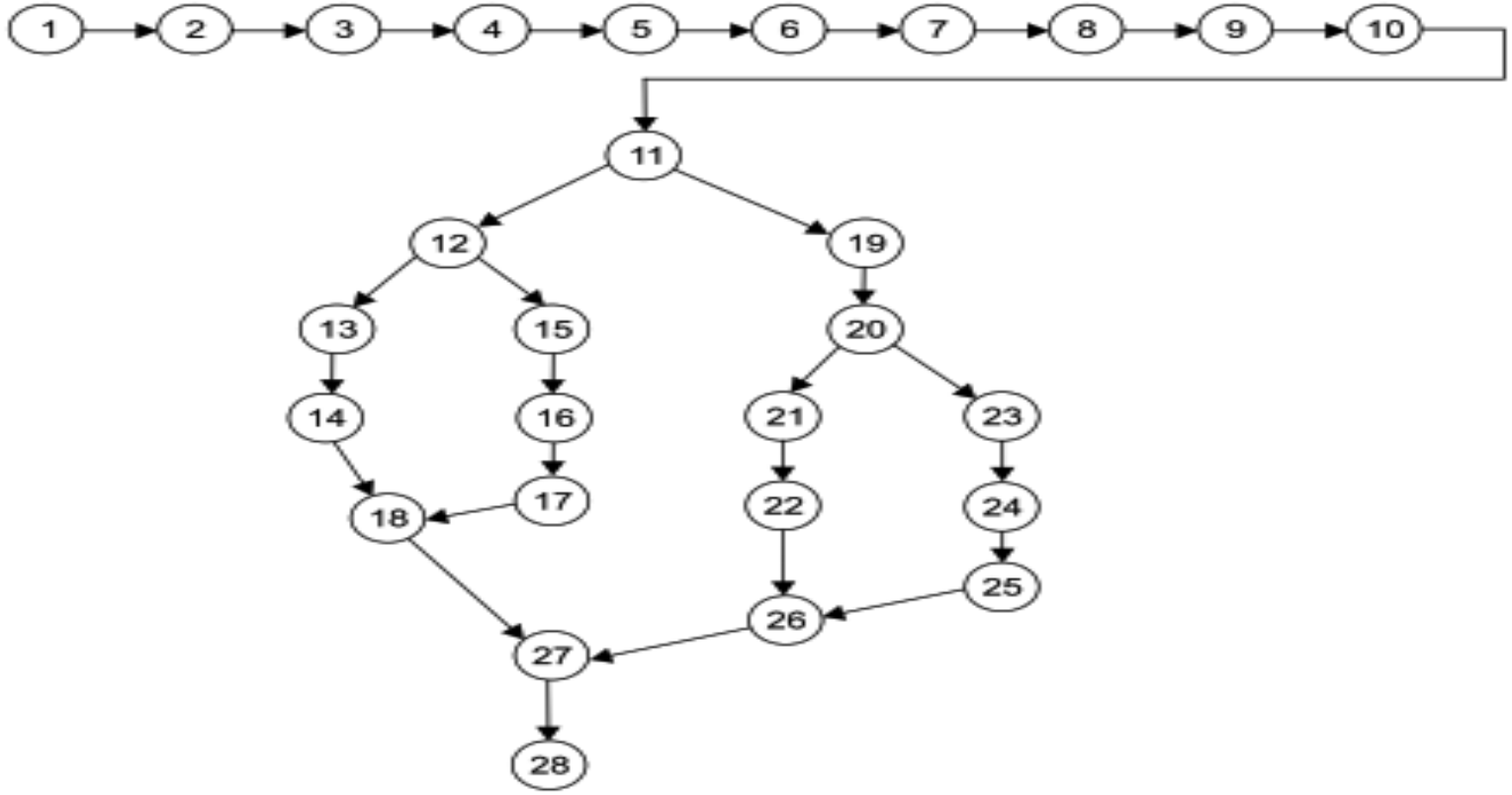
DATA FLOW TESTING LIMITATIONS

- Testers require good knowledge of programming.
- Time-consuming
- Costly process.

EXAMPLE: FIND THE LARGEST NUMBER AMONGST THREE NUMBERS.

```
#include<stdio.h>
#include<conio.h>
1.  void main()
2.  {
3.  float A,B,C;
4.  clrscr();
5.  printf("Enter number 1:\n");
6.  scanf("%f", &A);
7.  printf("Enter number 2:\n");
8.  scanf("%f", &B);
9.  printf("Enter number 3:\n");
10. scanf("%f", &C);
    /*Check for greatest of three numbers*/
11. if(A>B) {
12. if(A>C) {
13.         printf("The largest number is: %f\n",A);
14.         }
15. else  {
16.         printf("The largest number is: %f\n",C);
17.         }
18. }
19. else {
20. if(C>B) {
21.         printf("The largest number is: %f\n",C);
22.         }
23. else  {
24.         printf("The largest number is: %f\n",B);
25.         }
26. }
27. getch();
28. }
```

STEP I



STEP II & III

S. No.	Variable	Defined at node	Used at node
1.	A	6	11, 12, 13
2.	B	8	11, 20, 24
3.	C	10	12, 16, 20, 21

The du-paths with beginning node and end node are given as:

Variable	du-path (Begin, end)
A	6, 11 6, 12 6, 13
B	8, 11 8, 20 8, 24
C	10, 12 10, 16 10, 20 10, 21

TEST CASES

Test all du-paths					
S. No.	Inputs			Expected Output	Remarks
	A	B	C		
1.	9	8	7	9	6–11
2.	9	8	7	9	6–12
3.	9	8	7	9	6–13
4.	7	9	8	9	8–11
5.	7	9	8	9	8–11, 19, 20
6.	7	9	8	9	8–11, 19, 20, 23, 24
7.	8	7	9	9	10–12
8.	8	7	9	9	10–12, ,15, 16
9.	7	8	9	9	10, 11, 19, 20
10.	7	8	9	9	10, 11, 19–21

CLASS TASK

- Consider the following program:

```
static int find (int list[], int n, int key)
{
// binary search of ordered list
int lo = 0, mid;
int hi = n - 1;
int result = -1;
while ((hi >= lo) && (result == -1)) {
mid = (lo + hi) / 2;
if (list[mid] == key)
    result = mid;
else if (list[mid] > key)
    hi = mid - 1;
else // list[mid] < key
    lo = mid + 1;
}
return result;
}
```

ACTIVITIES

1. (10 Minutes) First individually construct the flow graph corresponding to this program.

2. (5 Minutes) Find a partner to work with in the group and check that you agree on the structure of the flow-graph for the program.

3. (10 minutes) For each variable, write down the < D, U > pairs.

4. (10 minutes) Write down tests that satisfy one of the following coverage criteria:

(a) All < D, U > pairs

(b) All < D, U > paths

5. (10 minutes) As a whole class, compare the tests sets devised for the two coverage criteria and discuss which is stronger. Can you think of a test that passes one of the two criteria but fails the other.

SLICE BASED TESTING

- We prepare various subsets (called slices) of a program with respect to its variables and their selected locations in the program.
- Each variable with one of its location will give us a program slice.
- A large program may have many smaller programs (its slices), each constructed for different variable subsets.
- "Program slicing is a technique for restricting the behaviour of a program to some specified subset of interest. A slice S(, n) of program P on variable , or set of variables, at statement n yields the portions of the program that contributed to the value of just before statement n is executed. S (, n) is called a slicing criteria.
- Slices can be computed automatically on source programs by analyzing data flow.
- Hence, slices are smaller than the original program and may be executed independently.
- Only two things are important here, variable and its selected location in the program.

GUIDELINES FOR SLICING

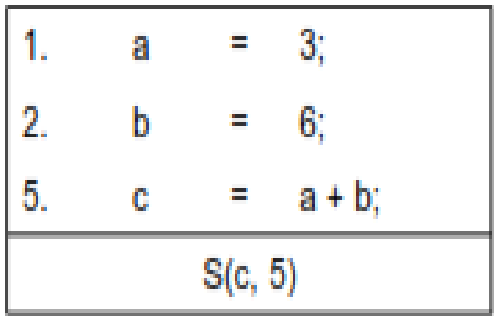
- All statements where variables are defined and redefined should be considered.
- All statements where variables receive values externally should be considered.
- All statements where output of a variable is printed should be considered.
- The status of all variables may be considered at the last statement of the program.

CREATION OF PROGRAM SLICES

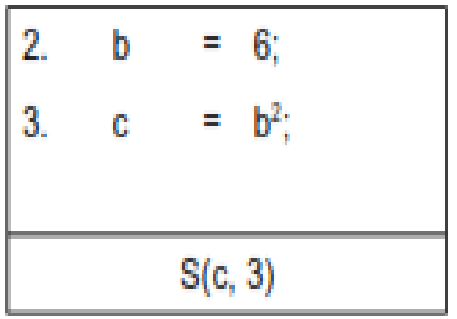
- Consider the portion of a program given in Figure 4.2 for the identification of its slices.
- a = 3;
- b = 6;
- c = b^2;
- d = a^2+ b^2;
- c = a + b;

PROGRAM SLICES

- We identify two slices for variable ‘c’ at statement number 3 and statement number 5.



Variable ‘c’ at statement 5



Variable ‘c’ at statement 5

EXAMPLE PROGRAM

- Consider the following program.

```
1. void main ( )
2. {
3. int a, b, c, d, e;
4. printf (“Enter the values of a, b and c \ n”);
5. scanf (“%d %d %d”, & a, &b, &c);
6. d = a+b;
7. e = b+c:
8. printf (“%d”, d);
9. printf (“%d”, e);
10. }
```

SOME SLICES

```
1.  main ( )
2.  {
3.  int a, b, c, d, e;
4.  printf ("Enter the values of a, b and c \ n");
5.  scanf ("%d %d %d", &a, &b, &c);
7.  e = b + c;
9.  printf ("%d", e);
10. }
```

Slice on criterion S (e, 10) = (1, 2, 3, 4, 5, 7, 9, 10)

```
1. main ( )
2. {
3. int a, b, c, d, e;
4. printf ("Enter the values of a, b and c \ n");
5. scanf ("%d %d %d", &a, &b, &c);
7.  e = b + c;
10. }
```

Slice on criterion S (e,7) = (1, 2, 3, 4, 5, 7,10)

```
1.  main ( )
2.  {
3.  int a, b, c, d, e;
4.  printf ("Enter the values of a, b and c \ n");
5.  scanf ("%d %d %d", &a, &b, &c);
10. }
```

Slice on criterion S (a, 5) = (1, 2, 3, 4, 5, 10)

```
1.  main ( )
2.  {
3.  int a, b, c, d, e;
4.  printf ("Enter the values of a, b and c \ n");
5.  scanf ("%d %d %d", &a, &b, &c);
6.  d = a + b;
8.  printf ("%d", d);
10. }
```

Slice on criterion S (d,10) = (1, 2, 3, 4, 5, 6, 8, 10)

```
1.  main ( )
2.  {
3.  int a, b, c, d, e;
4.  printf ("Enter the values of a, b and c \ n");
5.  scanf ("%d %d %d", &a, &b, &c);
6.  d = a + b;
10. }
```

Slice on criterion S (d,6) = (1, 2, 3, 4, 5, 6, 10)

GENERATION OF TEST CASES

- Every slice should be independently executable and may cover some lines of source code of the program.
- The test cases for the slices of the program given
- Generated slices to find the largest number amongst three numbers are S(A, 6), S(A, 13), S(A, 28), S(B, 8), S(B, 24), S(B, 28), S(C, 10), S(C, 16), S(C, 21), S(C, 28).

TEST CASES

Test cases using program slices of program to find the largest among three numbers

S. No.	Slice	Lines covered	A	B	C	Expected output
1.	S(A, 6)	1–6, 28	9			No output
2.	S(A, 13)	1–14, 18, 27, 28	9	8	7	9
3.	S(A, 28)	1–14, 18, 27, 28	8	8	7	9
4.	S(B, 8)	1–4, 7, 8, 28		9		No output
5.	S(B, 24)	1–11, 18–20, 22–28	7	9	8	9
6.	S(B, 28)	1–11, 19, 20, 23–28	7	9	8	9
7.	S(C, 10)	1–4, 9, 10, 28			9	No output
8.	S(C, 16)	1–12, 14–18, 27, 28	8	7	9	9
9.	S(C, 21)	1–11, 18–22, 26–28	7	8	9	9
10.	S(C, 28)	1–11, 18–22, 26–28	7	8	9	9

LIMITATIONS OF SLICE BASED TESTING

- Slice based testing is a popular structural testing technique and focuses on a portion of the program with respect to a variable location in any statement of the program.
- Hence slicing simplifies the way of testing a program's behavior with respect to a particular subset of its variables.
- But slicing cannot test a behavior which is not represented by a set of variables or a variable of the program.

That is all