Data Flow Testing

Why to we need Data Flow Testing?

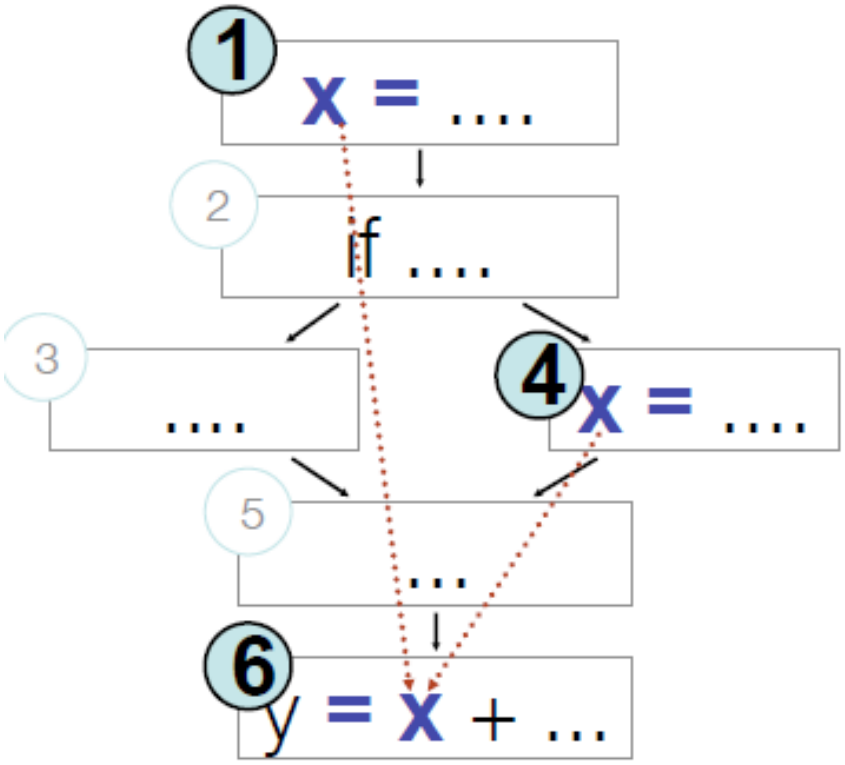
Statements interact through *data flow*

- Value computed in one statement, used in another
- Bad value computation revealed only when it is used

Basic idea

- Program paths along which variables are *defined* and then *used* should be covered

Data flow concept



- Value of x at 6 could be computed at 1 or at 4
- Bad computation at 1 or 4 could be revealed only if they are used at 6
- (1,6) and (4,6) are *def-use (DU) pairs*
  - defs at 1,4
  - use at 6

Variable Definition

A program variable is **DEFINED** when it appears:

- On the *left* hand side of an assignment statement *eg* **y = 17**
- in an input statement *eg* **read(y)**
- z=&x

…

Variable Use

A program variable is **USED** when it appears:

- on the *right* hand side of an assignment statement

eg **y = x+17**

- as an call-by-value parameter in a subroutine or function call

eg **y = sqrt(x)**

- in the predicate of a branch statement

eg **if (x > 0) { … }**

Variable Use

A variable can also be used and then re-defined in a single statement when it appears:

on *both* sides of an assignment statement

eg `y = y + 1 (y++)`

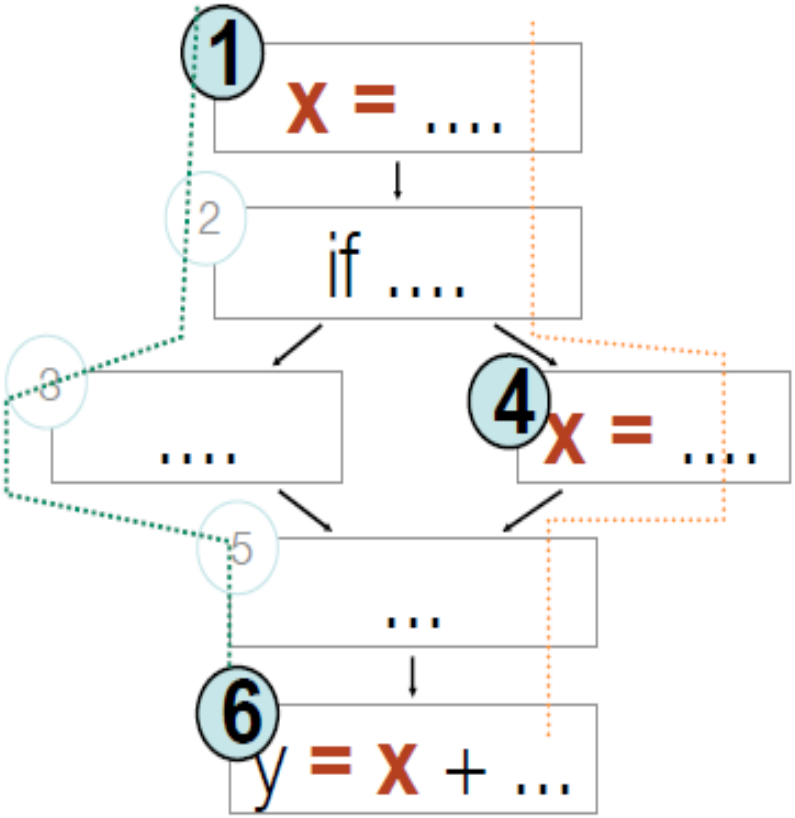
Variable Use: p-use and c-use

- Use in the predicate of a branch statement is a *predicate-use* **or "p-use"**
- Any other use is a *computation-use* **or "c-use"**
- For example, in the program fragment:

```
if ( x > 0 ) {
        print(y);
        }
```

there is a p-use of **x** and a c-use of **y**

Definition-Clear path (DU path)



- 1,2,3,5,6 is a definition-clear path from 1 to 6
  - x is not re-assigned between 1 and 6
- 1,2,4,5,6 is not a definition-clear path from 1 to 6
  - the value of x is "killed" (reassigned) at node 4
- (1,6) is a DU pair because 1,2,3,5,6 is a definition-clear path

More Dataflow Terms and Definitions

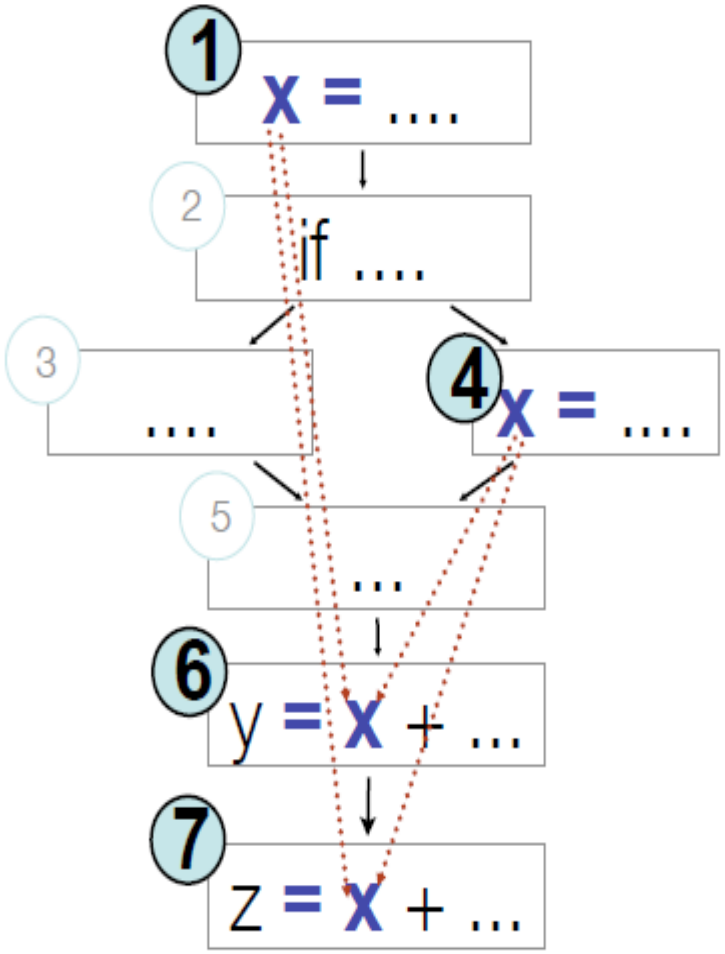
- A path is *definition clear* **("def-clear")** with respect to a variable **`v`** if it has no variable **re-**definition of **`v`** on the path.

- A *definition-use pair* **("du-pair")** with respect to a variable **`v`** is a double **`(d,u)`** such that
  - **`d`** is a node in the program's flow graph at which **`v`** is defined,
  - **`u`** is a node or edge at which **`v`** is used *and*
  - there is a def-clear path *with respect to* **`v`** from **`d`** to **`u`**

Adequacy criteria

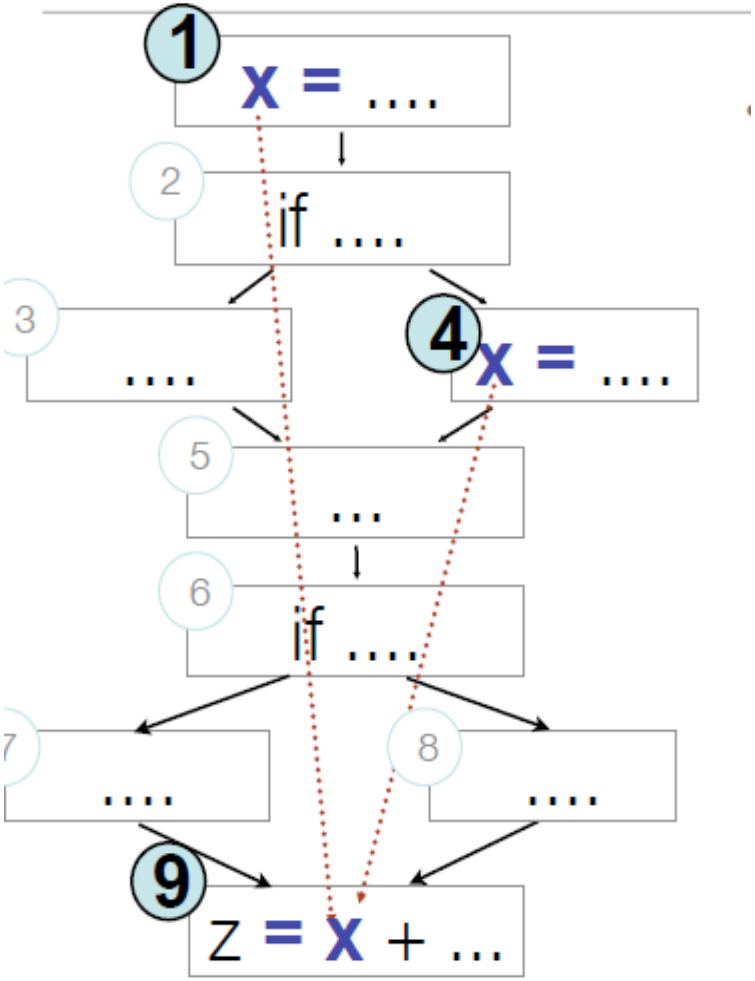
- All DU pairs: Each DU pair is exercised by at least one test case
- All DU paths: Each *simple* (non looping) DU path is exercised by at least one test case
- All definitions: For each definition, there is at least one test case which exercises a DU pair containing it

All DU pairs



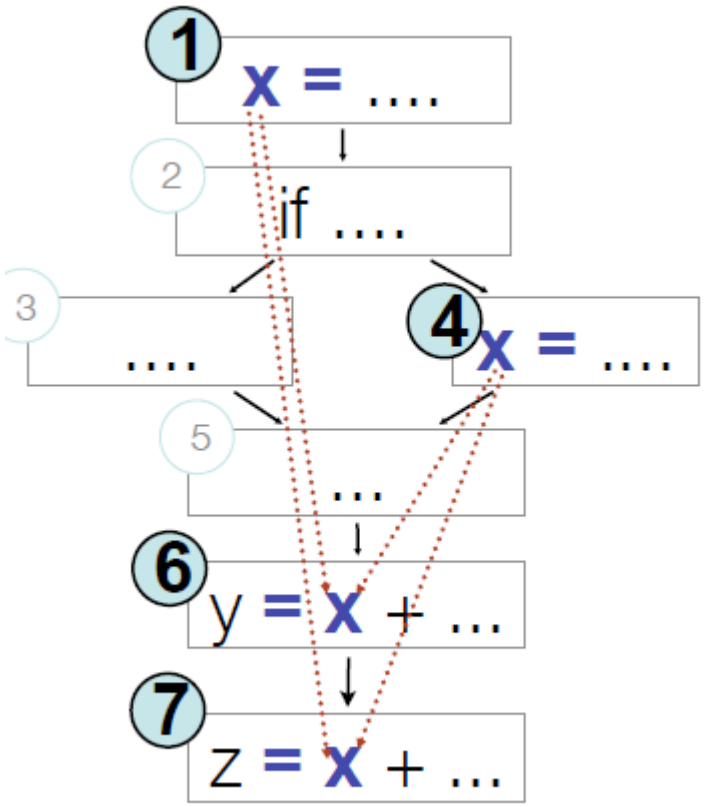
- Requires to cover all the following pairs:
  - def at 1 - use at 6
  - def at 1 - use at 7
  - def at 4 - use at 6
  - def at 4 - use at 7

All DU paths



- Requires to cover all the following pairs:
  - def at 1 - use at 9 (through 7)
  - def at 1 - use at 9 (through 8)
  - def at 4 - use at 9 (through 7)
  - def at 4 - use at 9 (through 8)

All definitions



- Requires to cover 2 pairs:
  - def at 1 - use at 6

    OR
  - def at 1 - use at 7

  - def at 4 - use at 6

    OR
  - def at 4 - use at 7

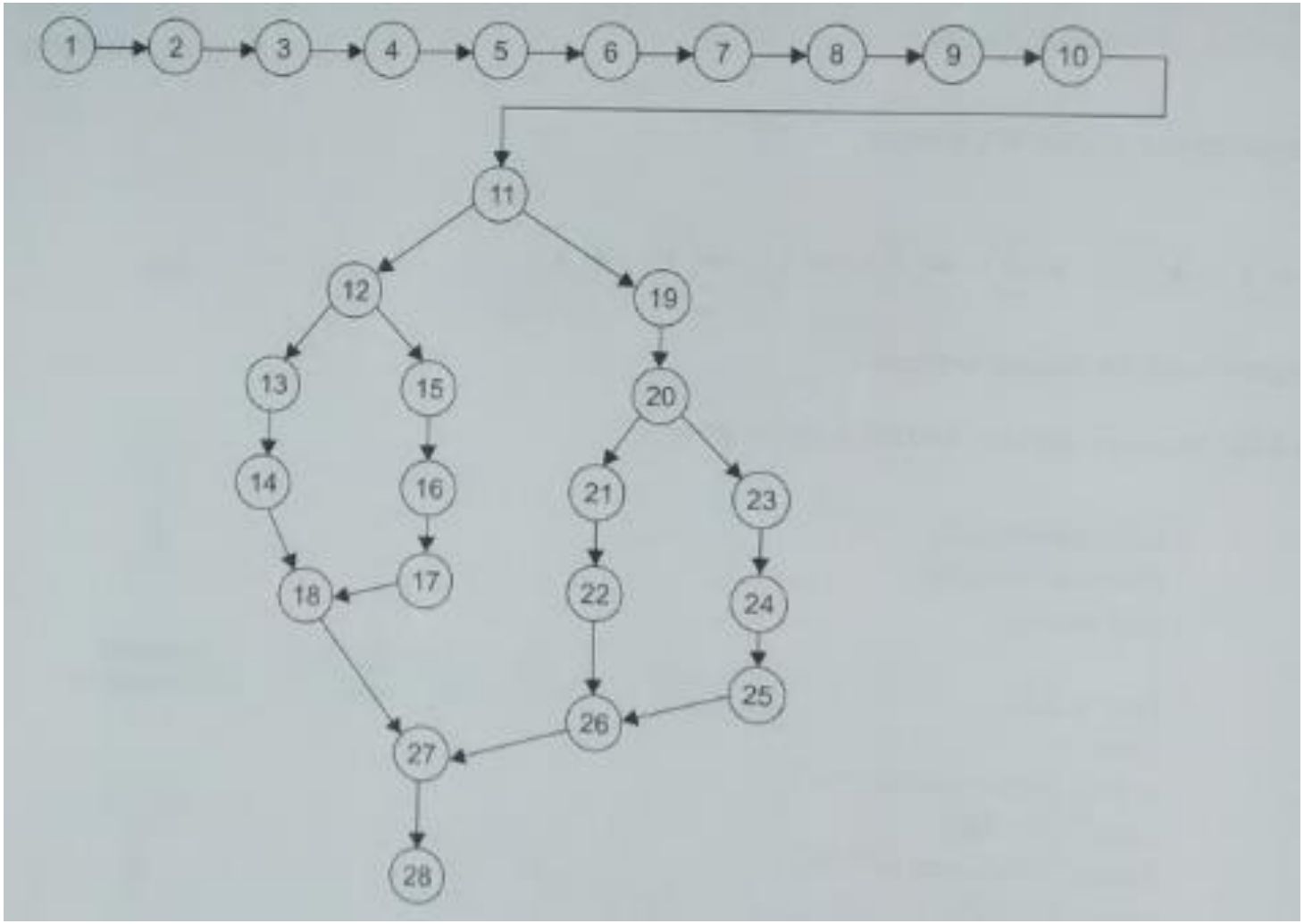
- For y def at 6
- For z def at 7

Ex#3:Program to find Largest of 3 numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
float A,B,C;
clrscr();
printf("Enter number 1:\n");
scanf("%f", &A);
printf("Enter number 2:\n");
scanf("%f", &B);
printf("Enter number 3:\n");
scanf("%f", &C);
/*Check for greatest of three numbers*/
if(A>B) {
if(A>C) {
        printf("The largest number is: %f\n",A);
        }
else  {
        printf("The largest number is: %f\n",C);
        }
}
else {
if(C>B) {
        printf("The largest number is: %f\n",C);
        }
```

```
else  {
        printf("The largest number is: %f\n",B);
        }
}
getch();
}
```

Program Flow graph



Define/ Use nodes and DU pairs

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

Variable	All possible pairs
A	6,11
	6,12
	6,13
B	8,11
	8,20
	8,24
C	10,12
	10,16
	10,20
	10,21

DU paths

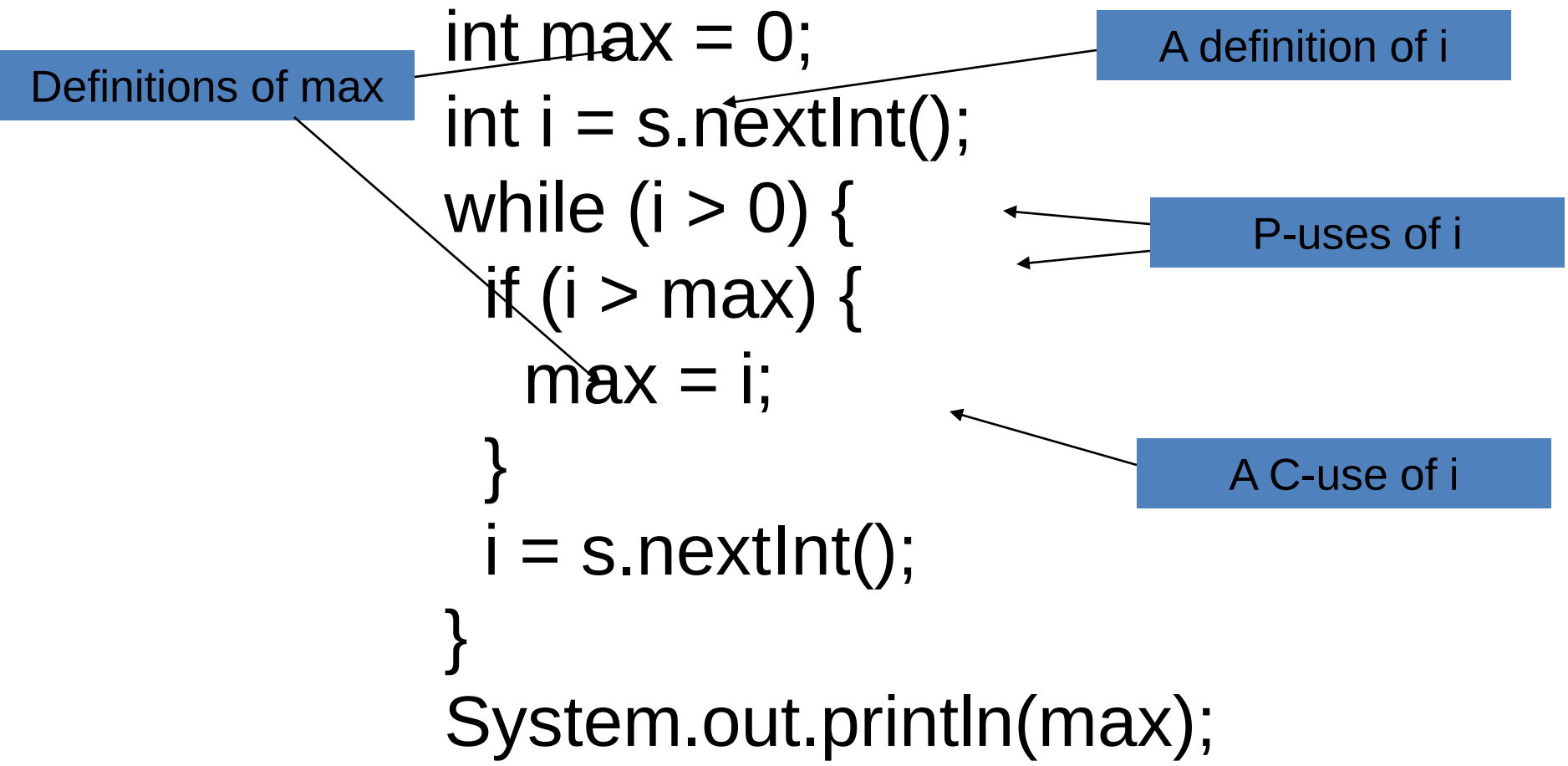
All possible pairs	Definition Clear?	DU Path
6,11	Yes	6-11
6,12	Yes	6-12
6,13	Yes	6-13
8,11	Yes	8-11
8,20	Yes	8-11,19,20
8,24	Yes	8-11,19,20,23,24
10,12	Yes	10-12
10,16	Yes	10-12, 15,16
10,20	Yes	10,11,19,20
10,21	Yes	10,11,19-21

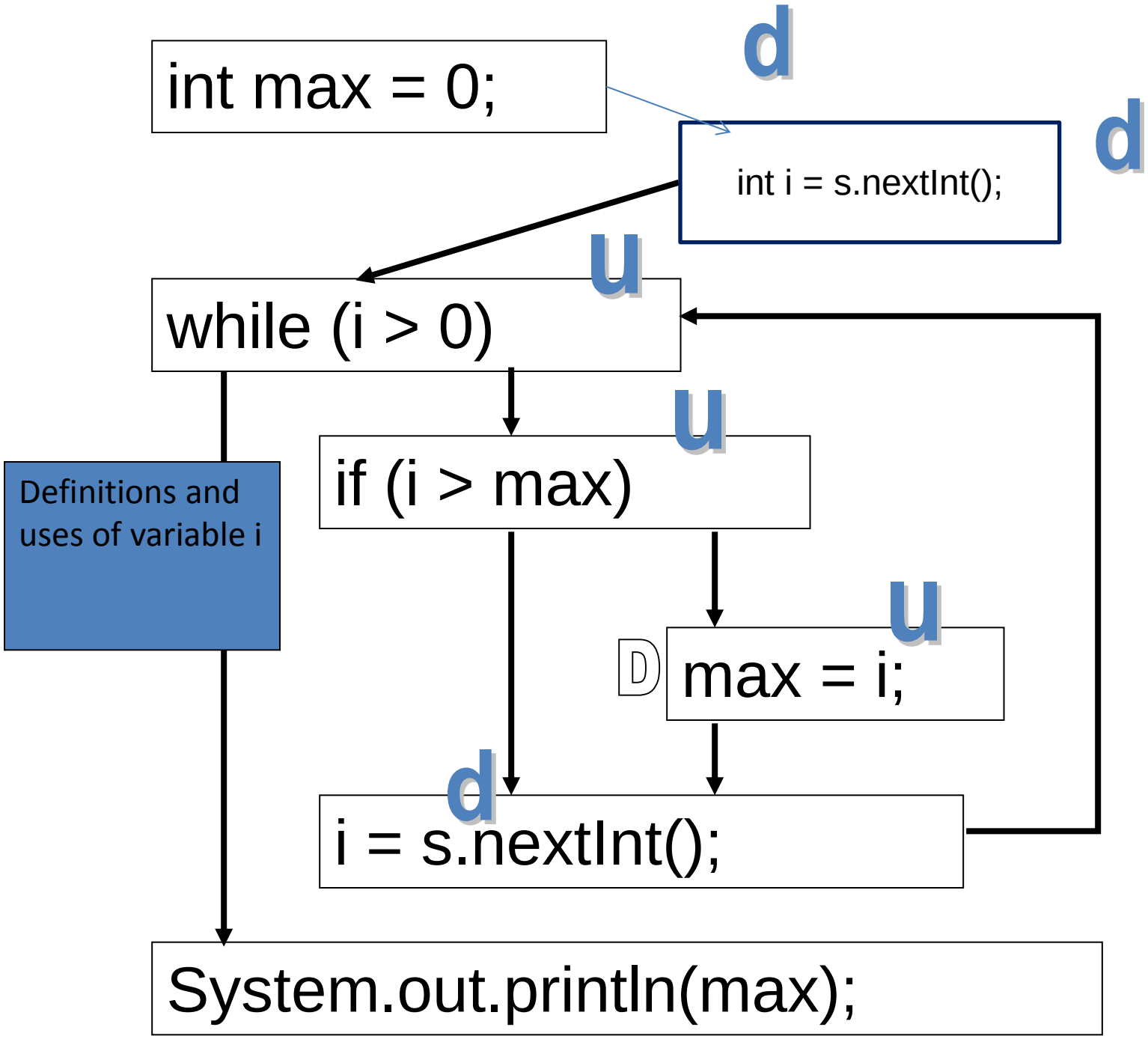
Test cases

Test all du-paths

S. No.	Inputs A	B	C	Expected Output	Remarks
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

Example#4


Definitions of max
A definition of i
P-uses of i
A C-use of i
int max = 0;
int i = s.nextInt();
while (i > 0) {
if (i > max) {
max = i;
}
i = s.nextInt();
}
System.out.println(max);


int max = 0;
d
int i = s.nextInt();
d
u
while (i > 0)
u
if (i > max)
Definitions and uses of variable i
D
max = i;
u
d
i = s.nextInt();
System.out.println(max);

Example#5

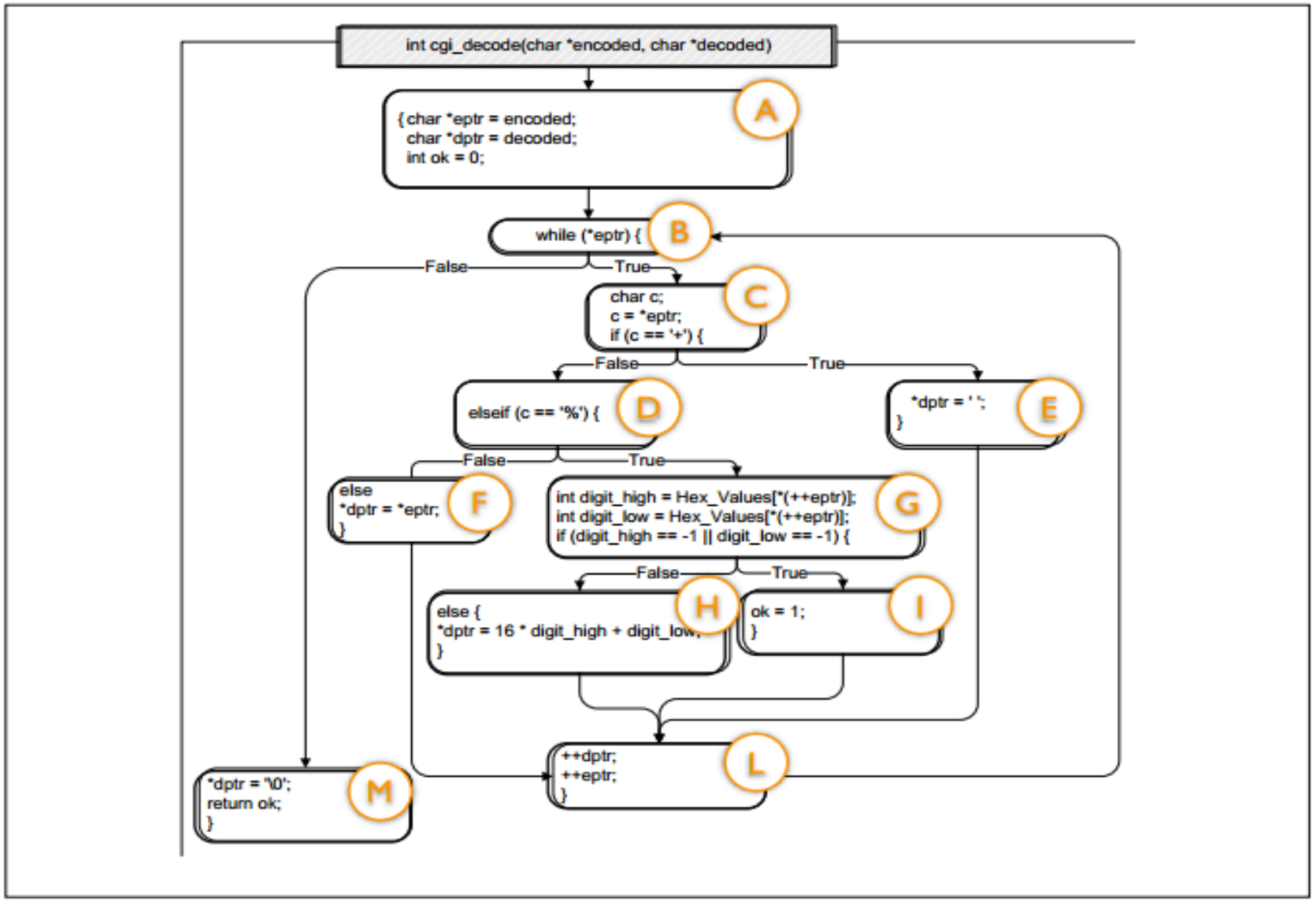
```
1 Main
2 float A,B,C,D, x1, x2
3 boolean is_complex
4 input(A,B,C)
5 D = B*B -4*A*C
6 if D < 0.0
7 then      is_complex = true
8 else  is_complex=false
9 end if
10 if not is_complex
11 then x1 = (-B + sqrt(D))/(2.0*A)
12              x2 = (-B - sqrt(D))/(2.0*A)
13 end if
14 output(x1,x2)
15 end
```

var	def	use
A	2,4	5,11,12
B	2,4	5,11,12
C	2,4	5
D	2,5	6,11,12
x1	2,11	14
x2	2,12	14
is_complex	3,7,8	10

var	all pairs	def-clear	du-path
A,B	(2,5)	N	
	(2,11)	N	
	(2,12)	N	
	(4,5)	Y	4,5
	(4,11)	Y	4,5,6,7,9,10,11
			4,5,6,8,9,10,11
	(4,12)	Y	4,5,6,7,9,10,11,12
			4,5,6,8,9,10,11,12
C	(2,5)	N	
	(4,5)	Y	4,5

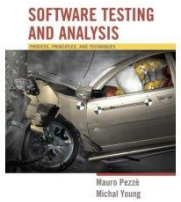
```
14  int cgi_decode(char *encoded, char *decoded) {
15    char *eptr = encoded;
16    char *dptr = decoded;
17    int ok=0;
18    while (*eptr) {
19      char c;
20      c = *eptr;
21
22      if (c == '+') {   /* Case 1: '+' maps to blank */
23        *dptr = ' ';
24      } else if (c == '%') { /* Case 2: '%xx' is hex for character xx */
25        int   digit_high = Hex_Values[*(++eptr)];
26        int   digit_low  = Hex_Values[*(++eptr)];
27        if (    digit_high == -1 || digit_low == -1 ) {
28          /* *dptr='?'; */
29          ok=1; /* Bad return code */
30        } else {
31          *dptr = 16* digit_high + digit_low;
32        }
33      } else { /* Case 3: All other characters map to themselves */
34        *dptr = *eptr;
35      }
36      ++dptr;
37      ++eptr;
38    }
39    *dptr = '\0';                    /* Null terminator for string */
40    return ok;
41  }
```

Example 2-CGI decode



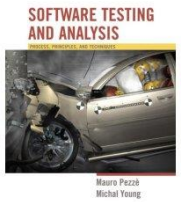
Definitions and Uses

Variable	Definitions	Uses
encoded	14	15
decoded	14	16
*eptr	15, 25, 26, 37	18, 20, 25, 26, 34
eptr	15, 25, 26, 37	15, 18, 20, 25, 26, 34, 37
*dptr	16, 23, 31, 34, 36, 39	
dptr	16 36	16, 23, 31, 34, 36, 39
ok	17, 29	40
c	20	22, 24
digit_high	25	27, 31
digit_low	26	27, 31
Hex_Values	–	25, 26



DU Pairs

Variable	DU Pairs
*eptr	⟨ 15, 18 ⟩, ⟨ 15, 20 ⟩, ⟨ 15, 25 ⟩, ⟨ 15, 34 ⟩ (25,25) (26,26) ⟨ 37, 18 ⟩, ⟨ 37, 20 ⟩, ⟨ 37, 25 ⟩, ⟨ 37, 34 ⟩
eptr	⟨ 15, 15 ⟩, ⟨ 15, 18 ⟩, ⟨ 15, 20 ⟩, ⟨ 15, 25 ⟩, ⟨ 15, 34 ⟩, ⟨ 15, 37 ⟩, (25,25) (26,26) ⟨ 25, 26 ⟩, ⟨ 26, 37 ⟩ ⟨ 37, 18 ⟩, ⟨ 37, 20 ⟩, ⟨ 37, 25 ⟩, ⟨ 37, 34 ⟩, ⟨ 37, 37 ⟩
*dptr	
dptr	⟨ 16, 16 ⟩, ⟨ 16, 23 ⟩, ⟨ 16, 31 ⟩, ⟨ 16, 34 ⟩, ⟨ 16, 36 ⟩, ⟨ 16, 39 ⟩, ⟨ 36, 23 ⟩, ⟨ 36, 31 ⟩, ⟨ 36, 34 ⟩, ⟨ 36, 36 ⟩, ⟨ 36, 39 ⟩
ok	⟨ 17, 40 ⟩, ⟨ 29, 40 ⟩
c	⟨ 20, 22 ⟩, ⟨ 20 24 ⟩
digit_high	⟨ 25, 27 ⟩, ⟨ 25, 31 ⟩
digit_low	⟨ 26, 27 ⟩, ⟨ 26, 31 ⟩
encoded	⟨ 14, 15⟩
decoded	⟨ 14, 16⟩



Data flow diagrams

- Refer Notes