Data Flow Testing

Define/Reference Anomalies

- A variable is defined but never used / referenced.
  - def calculate_total(price, tax):
  - total = price + tax # 'total' is defined
  - return tax # 'total' is never used or referenced
- A variable is used but never defined.
  - def display_message():
  - print(message) # 'message' is used here
  - message = "Hello, world!"
- A variable is defined twice before it is used.
  - def set_status(status_code):
  - current_status = "pending" # First definition
  - current_status = "active" # Second definition, overwriting the first
  - print(current_status) # 'current_status' is used here
- A variable is used before even first-definition.
  - def process_data(data):
  - # The programmer mistakenly uses 'data_processed' here,
  - # before it has been defined.
  - processed_result = data_processed * 2
  -
  - # The definition happens later.
  - data_processed = data + 10

Terms

- Definition (d): A point in the code where a variable is assigned a value (e.g., x = 5;).
- Use (u): A point where a variable's value is read or referenced. There are two types:
  - C-use (Computation Use):Used in a calculation or expression. Example: y = x +1.
  - P-use (Predicate Use): Used in a logical condition that influences control flow. Example: if (x > 5).
- Undefined (a): The state of a variable after it is declared but before it is assigned a value.
  - int myNumber; // 'myNumber' is in the undefined state

Definitions

- 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.
    - C-Use
    - P-Use

Definitions

- Definition use Path
  - A definition use path (denoted as du-path) for a variable ‘v ’ is a path between two nodes ‘m’ and ‘n’ where ‘m’ is the initial node in the path but the defining node for variable ‘v ’ (denoted as DEF (v, m)) and ‘n’ is the final node in the path but usage node for variable ‘ v’ (denoted as USE (v, n)).
- Definition clear path
  - A definition clear path (denoted as dc-path) for a variable ‘ v’ is a definition use path with initial and final nodes DEF (v, m) and USE (v , n) such that no other node in the path is a defining node of variable ‘v ’.
- The du-paths that are not definition clear paths are potential troublesome paths.

Identification of du and dc Paths

The various steps for the identification of du and dc paths are given as:

(i) Draw the program graph of the program.

(ii) 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

(iii) Generate all du-paths from define/use variable table of step (iii) using the following format:

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

(iv) Identify those du-paths which are not dc-paths.

Testing Strategies Using du-Paths

- 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.
- Test all definitions
  - Find paths from every definition of every variable to at least one use of that variable; we may choose any strategy for testing.

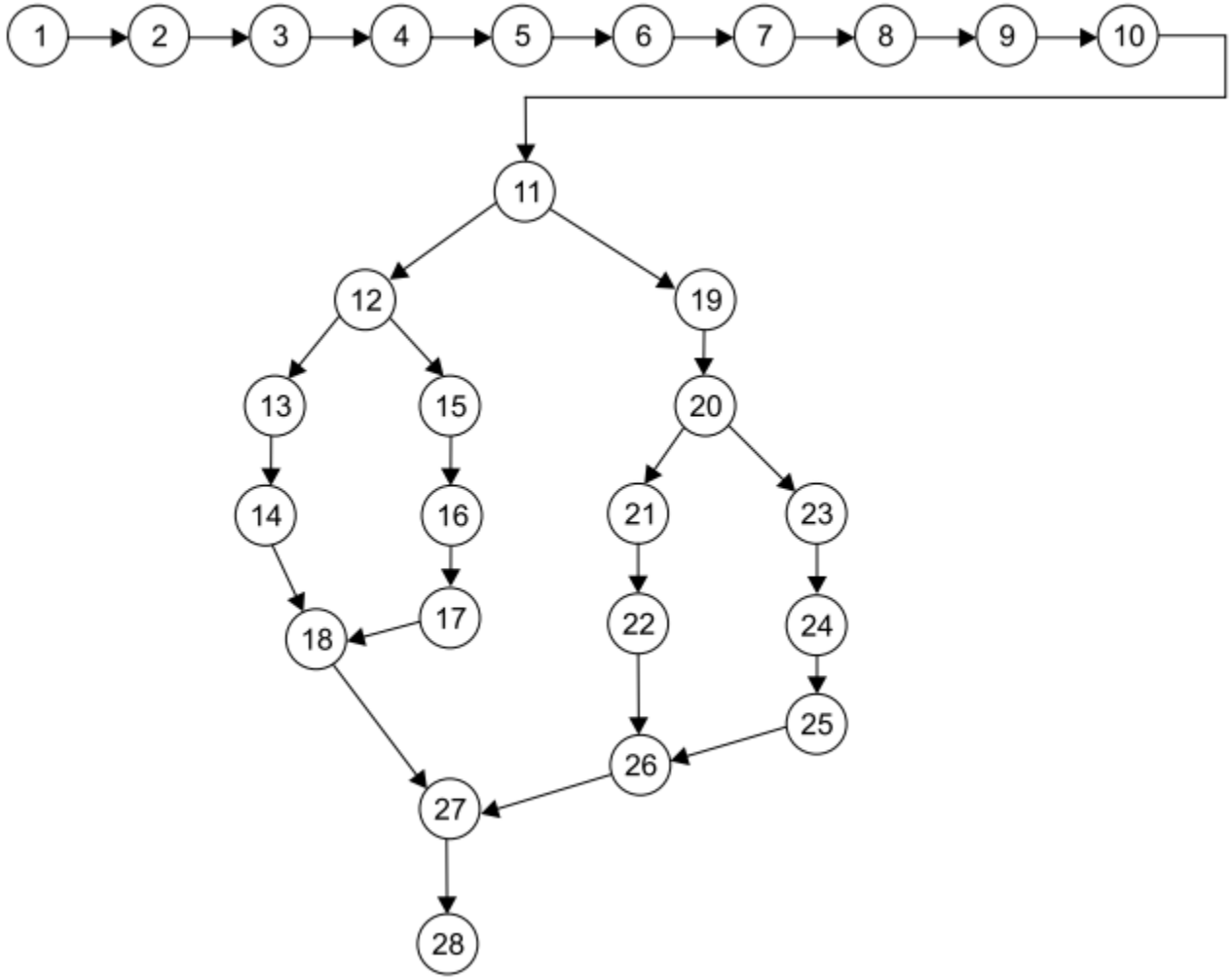
Generation of Test Cases

```
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.           }
```

(*Contd.*)

```
23.           else  {
24.                   printf("The largest number is: %f\n",B);
25.                   }
26.   }
27.   getch();
28.   }
```

Generation of Test Cases



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

	Paths	Definition clear?
All du paths and all uses (Both are same in this example)	6–11	Yes
	6–12	Yes
	6–13	Yes
	8–11	Yes
	8–11, 19, 20	Yes
	8–11, 19, 20, 23, 24	Yes
	10–12	Yes
	10–12, 15, 16	Yes
	10, 11, 19, 20	Yes
	10, 11, 19–21	Yes
All definitions	6–11	Yes
	8–11	Yes
	10–12	Yes

Here all du-paths and all-uses paths are the same (10 du-paths). But in the 3rd case, for all definitions, there are three paths.

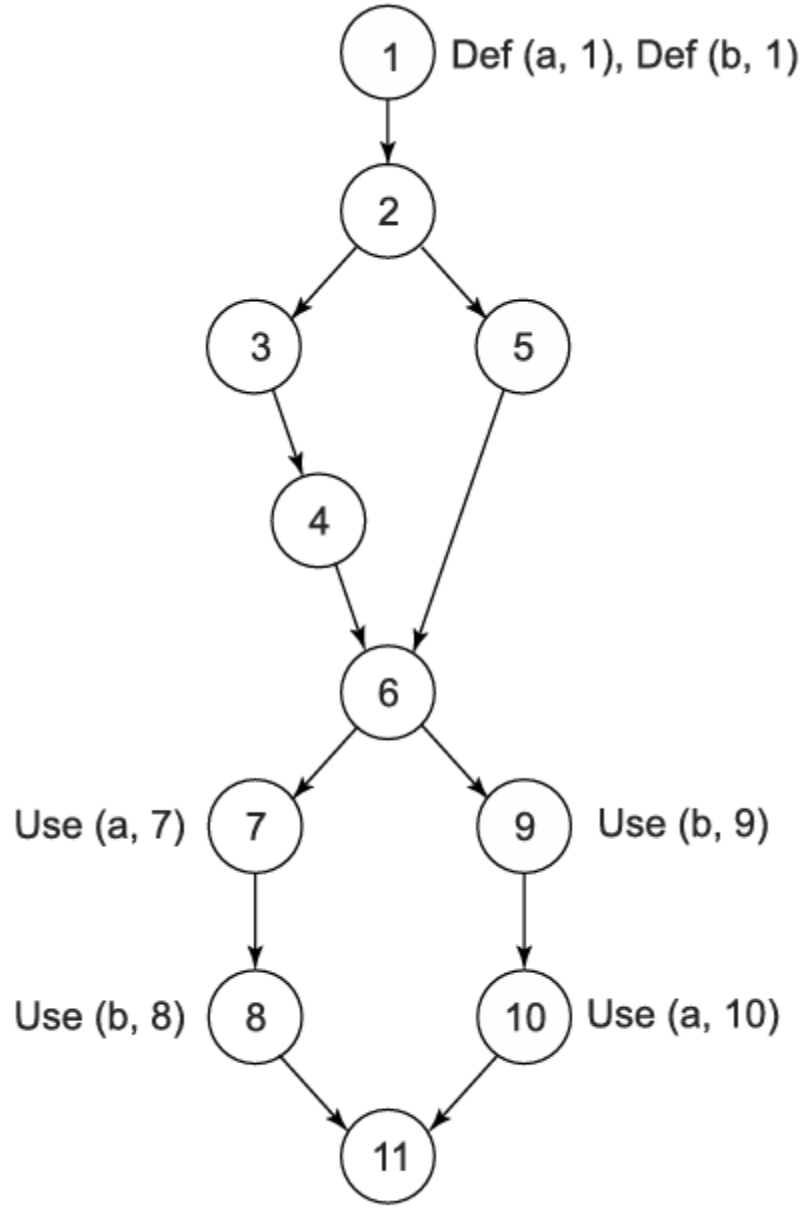
Test all du-paths

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

Test All definitions

S. No.	Inputs			Expected Output	Remarks
	A	B	C		
1.	9	8	7	9	6–11
2.	7	9	8	9	8–11
3.	8	7	9	9	10–12

- In this example all du-paths and all uses yield the same number of paths.
- This may not always be true. If we consider the following graph and find du paths with all three strategies, we will get a different number of all-du paths and all-uses paths.



Def/Use nodes table

S. No.	Variables	Defined at node	Used at node
1.	a	1	7, 10
2.	b	1	8, 9

The du paths are identified as:

S. No.	Variables	du-paths (Begin, end)
1.	a	1, 7 1, 10
2.	b	1, 8 1, 9

	Paths	Definition clear?
All du paths	1–4, 6, 7	Yes
(8 paths)	1, 2, 5–7	Yes
	1–4, 6, 9, 10	Yes
	1, 2, 5, 6, 9, 10	Yes
	1–4, 6, 7, 8	Yes
	1, 2, 5–8	Yes
	1–4, 6, 9	Yes
	1, 2, 5, 6, 9	Yes

(*Contd.*)

	Paths	Definition clear?
All uses	1–4, 6, 7	Yes
(4 paths)	1–4, 6, 9, 10	Yes
	1–4, 6-8	Yes
	1–4, 6, 9	Yes
All definitions	1–4, 6, 7	Yes
(2 paths)	1–4, 6–8	Yes