SENG 275

SOFTWARE TESTING

DR. NAVNEET KAUR POPLI

DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING

University of Victoria

BRANCH/DECISION COVERAGE

University of Victoria

Branch/Decision coverage

- Complex programs often rely on lots of **complex conditions** (e.g., if statements composed of many conditions). When testing these programs, aiming at 100% line or block coverage might not be enough to **cover all the cases** we want. We need a stronger criterion.
- Branch coverage (or decision coverage) works similar to line and statement coverage, except with branch coverage we count (or aim at covering) **all the possible decision outcomes**.
- A test suite will achieve 100% branch (or decision) coverage when tests exercise all the possible outcomes of decision blocks.

Branch/Decision coverage

- Decisions (or branches) are easy to identify in a CFG. Arrows with either true or false (i.e., both the arrows going out of a decision block) are branches, and therefore must be exercised.

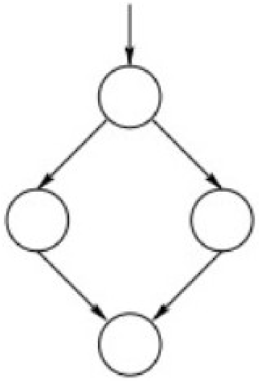
$$\text{branch coverage} = \frac{\text{decision outcomes covered}}{\text{decision outcomes total}} \cdot 100$$

Branch/Decision coverage

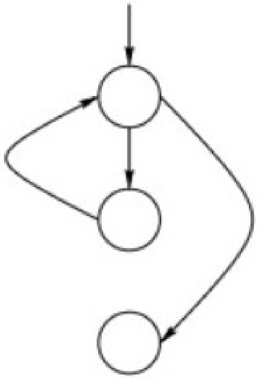
- **Test requirements**: branches in the program.
- **Coverage measure**: number of executed branches/total number of branches
- A **control-flow graph** (or CFG) is a representation of all paths that might be traversed during the execution of a piece of code. It consists of *basic blocks*, *decision blocks*, and *arrows/edges* that connect these blocks.

Control Flow Graphs

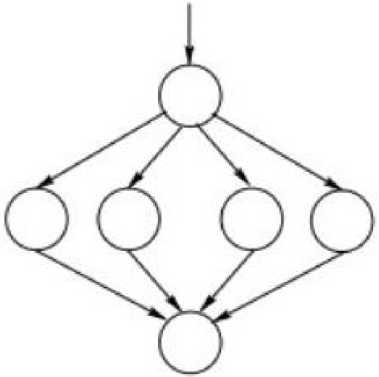
- Shows all execution paths a program *might* take
- Trace execution without executing program
- Nodes → Basic blocks
- Transitions → Control transfers



If-then-else



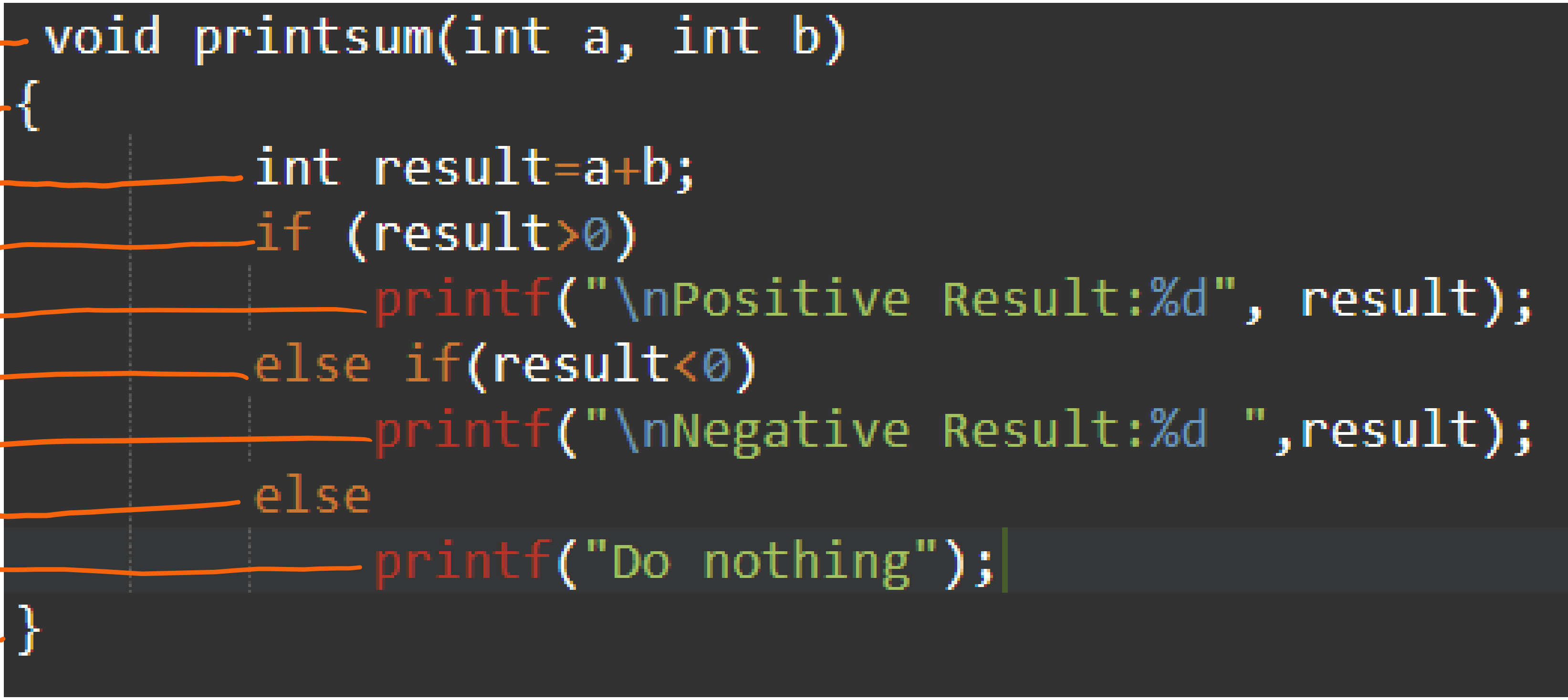
while



case

https://dzone.com/articles/how-draw-control-flow-graph

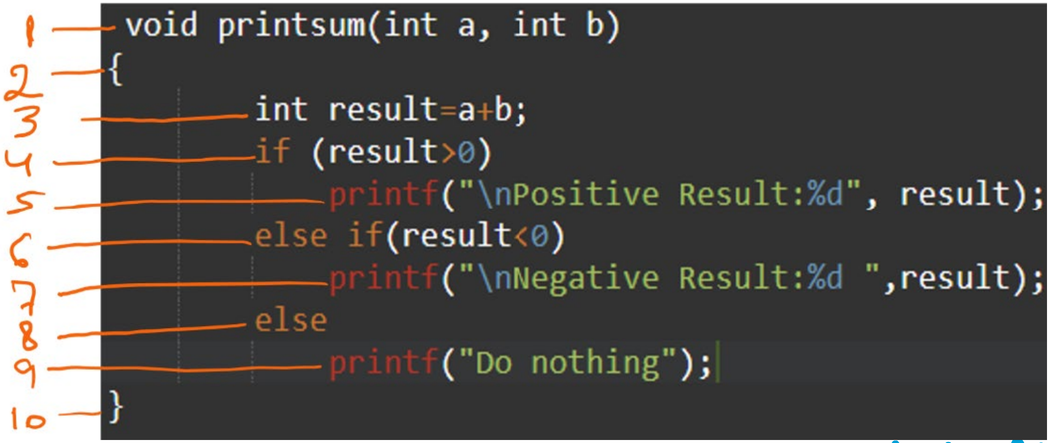
Let’s add another ‘else’ to pur previous program



```
1   void printsum(int a, int b)
2   {
3           int result=a+b;
4           if (result>0)
5                   printf("\nPositive Result:%d", result);
6           else if(result<0)
7                   printf("\nNegative Result:%d ",result);
8           else
9                   printf("Do nothing");
10  }
```

- Line 8 and 9 are never covered by either of the two test cases, (5,4), (-5,-4).
- So you need another test case(0,0).

Test cases



```
1   void printsum(int a, int b)
2   {
3           int result=a+b;
4           if (result>0)
5                   printf("\nPositive Result:%d", result);
6           else if(result<0)
7                   printf("\nNegative Result:%d ",result);
8           else
9                   printf("Do nothing");
10  }
```

Test Case #1

a=3, b=9

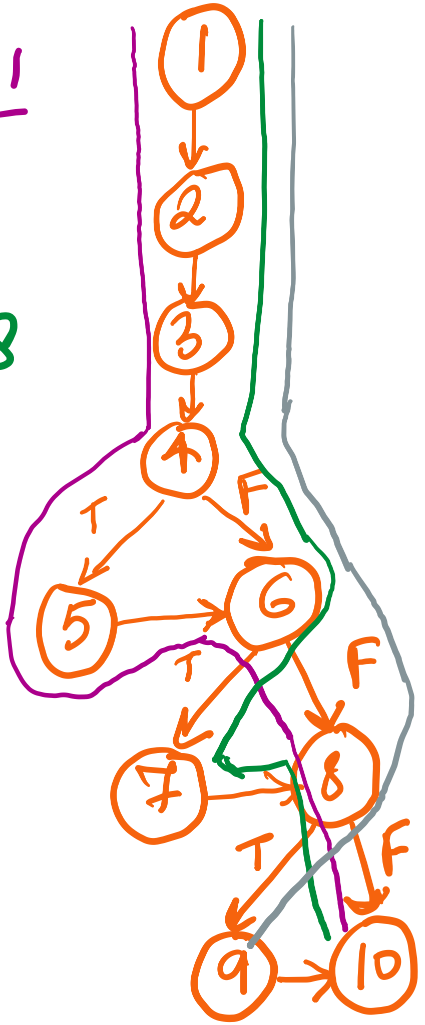
Test Case #2

a=-5, b=-8

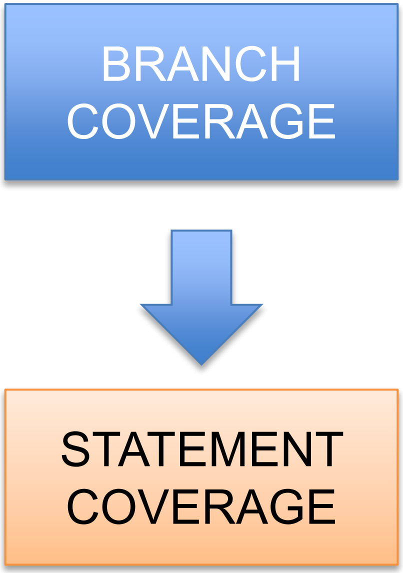
Test Case #3

a=0, b=0

These 3 test cases in totality give 100% Branch Coverage



Test criteria subsumption

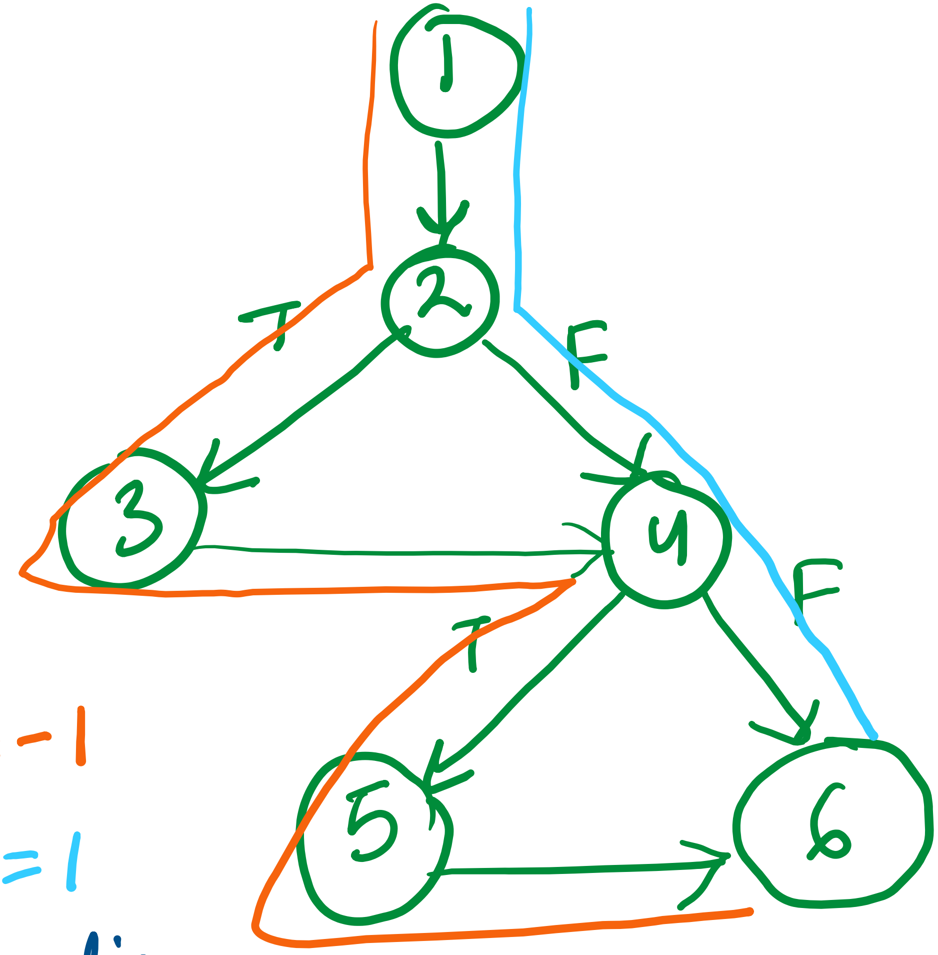


Given the following code, create a CFG and provide test inputs for achieving 100% statement and branch coverage

```
function inc(p, q) {
    if(q == 0)
        q=1;
    if( p < 0 )
        p = -p;
    return p + q/q; }
```

Given the following code, provide test inputs for achieving 100% statement and branch coverage

```
1.function inc(p, q) {
2.   if(q == 0)
3.         q=1;
4.   if( p < 0 )
5.         p = -p;
6.   return p + q/q; }
```



Test Case 1: q = 0 , p = -1

Test Case 2: q = 1 , p = 1

100% Branch coverage implies 100% Statement coverage

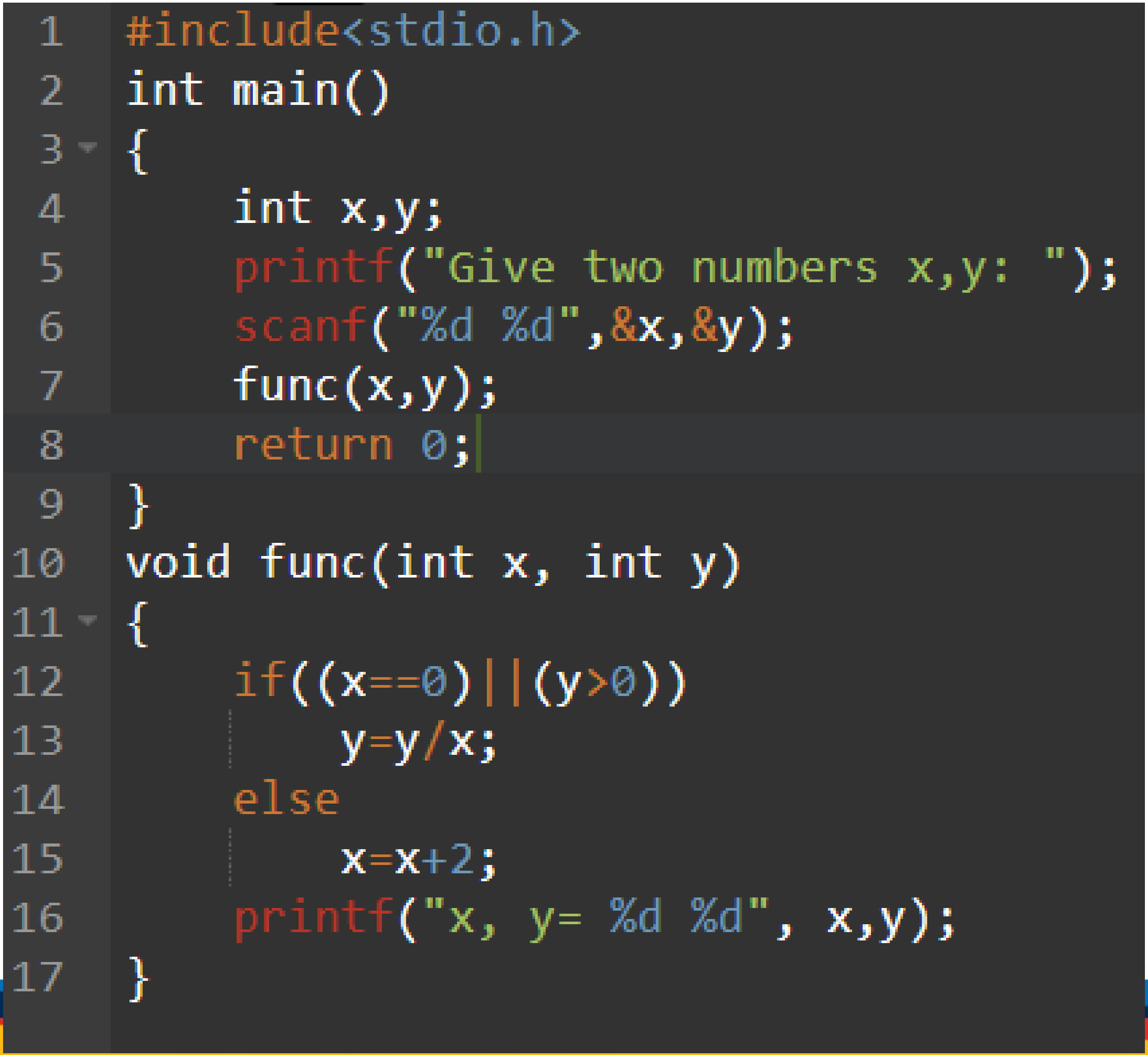
IS BRANCH COVERAGE ENOUGH ?

IS BRANCH COVERAGE ENOUGH ?

NO ! WE NEED BOTH BRANCH AND CONDITION COVERAGE

Let’s consider a function that you need to cover

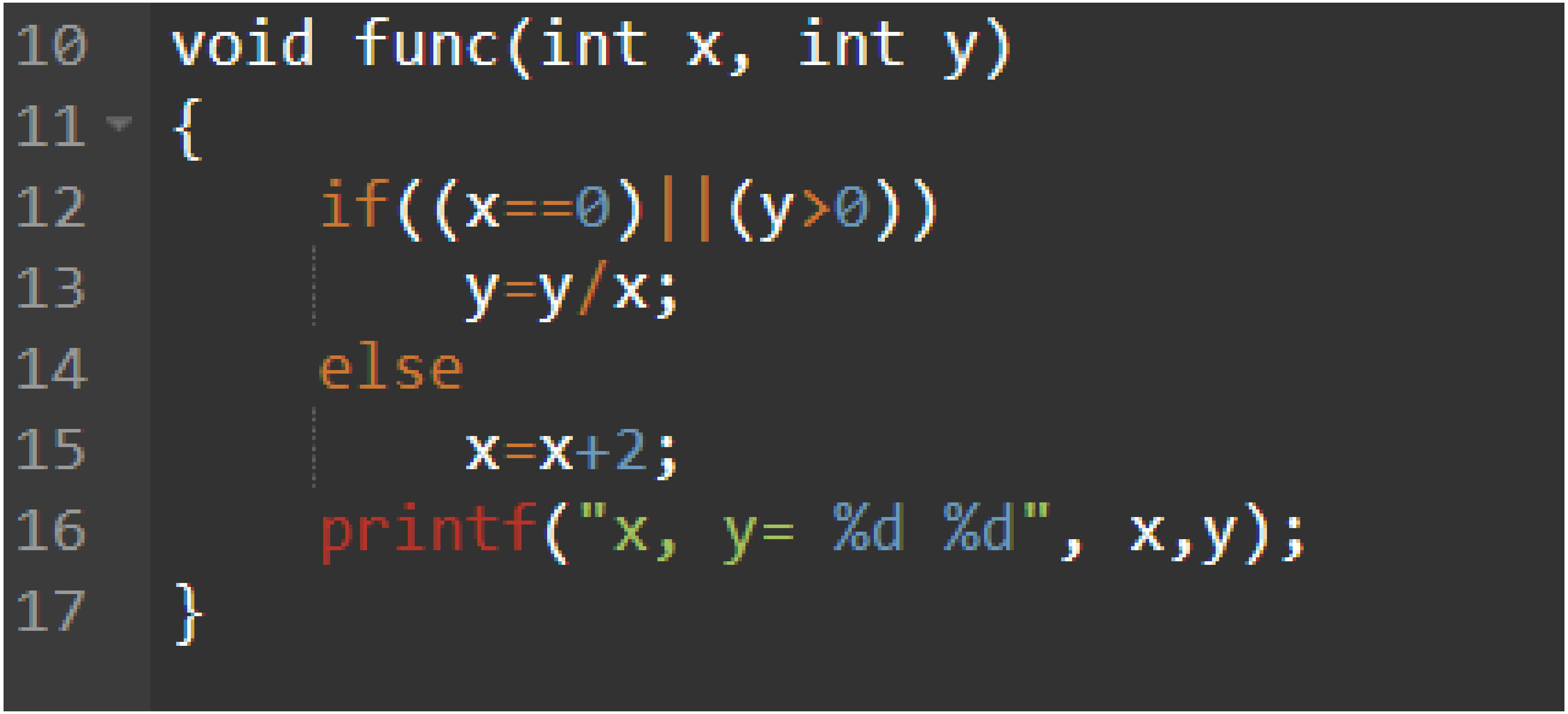
- Take two integers x, y. if x==0 or y>0, y=y/x. else x=x+2. print x and y



```c
#include<stdio.h>
int main()
{
    int x,y;
    printf("Give two numbers x,y: ");
    scanf("%d %d",&x,&y);
    func(x,y);
    return 0;
}
void func(int x, int y)
{
    if((x==0)||(y>0))
        y=y/x;
    else
        x=x+2;
    printf("x, y= %d %d", x,y);
}
```

Let’s consider a function that you need to cover

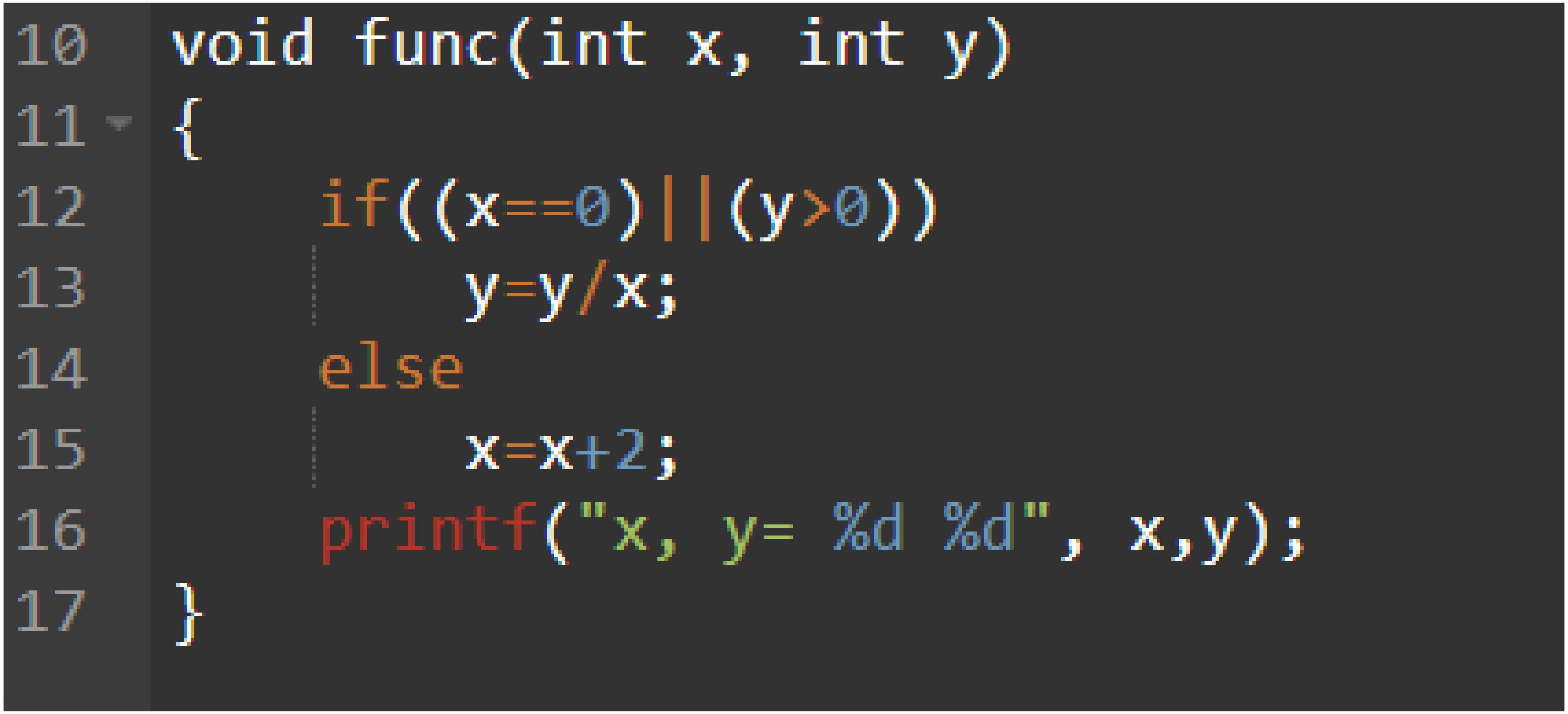
- Create test cases for 100% branch coverage.



```c
void func(int x, int y)
{
    if((x==0)||(y>0))
        y=y/x;
    else
        x=x+2;
    printf("x, y= %d %d", x,y);
}
```

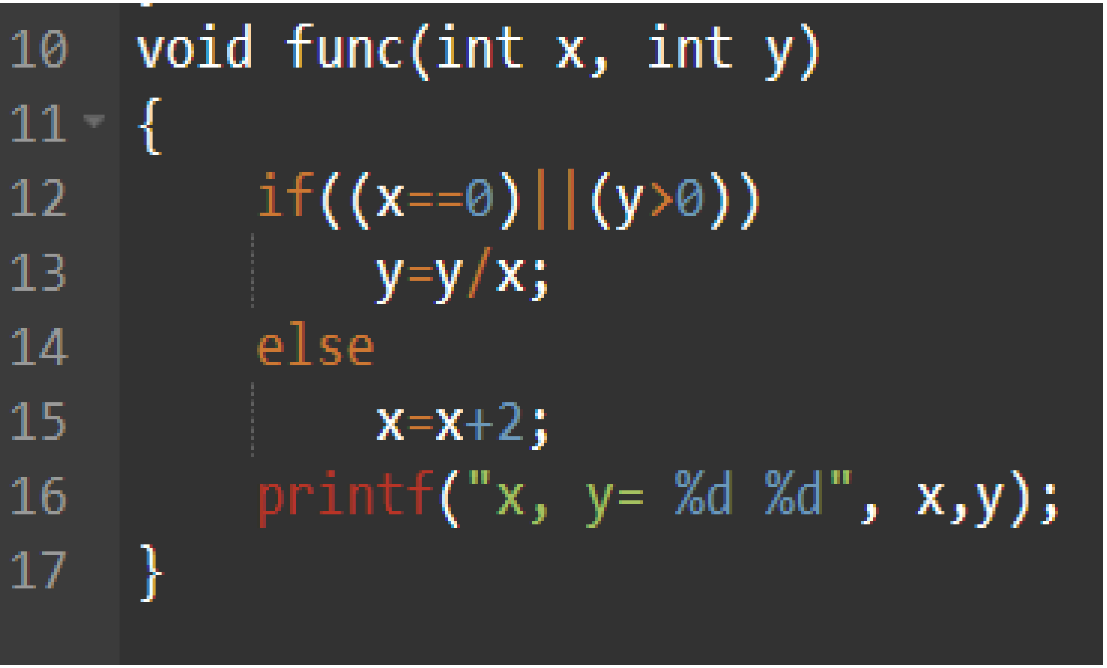
Let’s consider a function that you need to cover

- What happens if we take test cases (x,y) as (5,5) and (5,-5) ?
- Did you achieve 100% branch coverage?
- Is it enough?

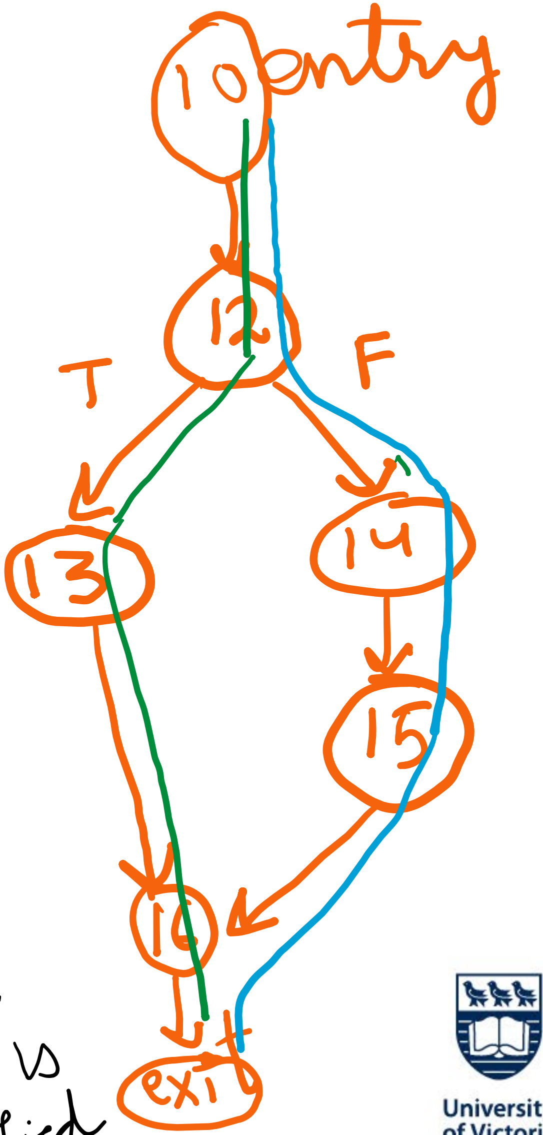


```c
void func(int x, int y)
{
    if((x==0)||(y>0))
        y=y/x;
    else
        x=x+2;
    printf("x, y= %d %d", x,y);
}
```

Create a CFG for the function



```
10  void func(int x, int y)
11  {
12      if((x==0)||(y>0))
13          y=y/x;
14      else
15          x=x+2;
16      printf("x, y= %d %d", x,y);
17  }
```

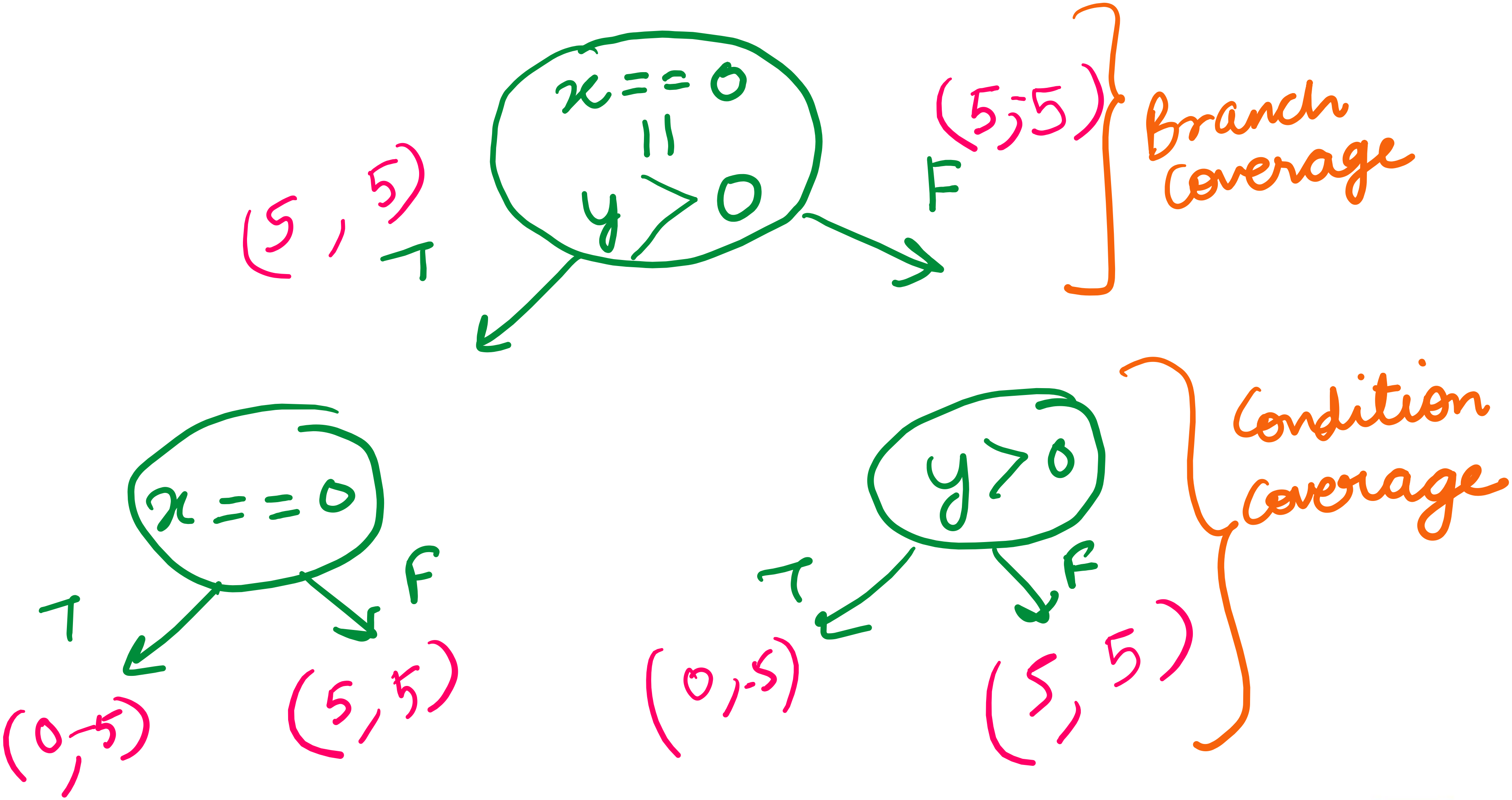


Test 1: $x = 0, y=5$

Test 2: $x=5, y=0$

So we achieve a 100% Branch Coverage with these two test cases. But error $\boxed{y=y/x}$ is not identified

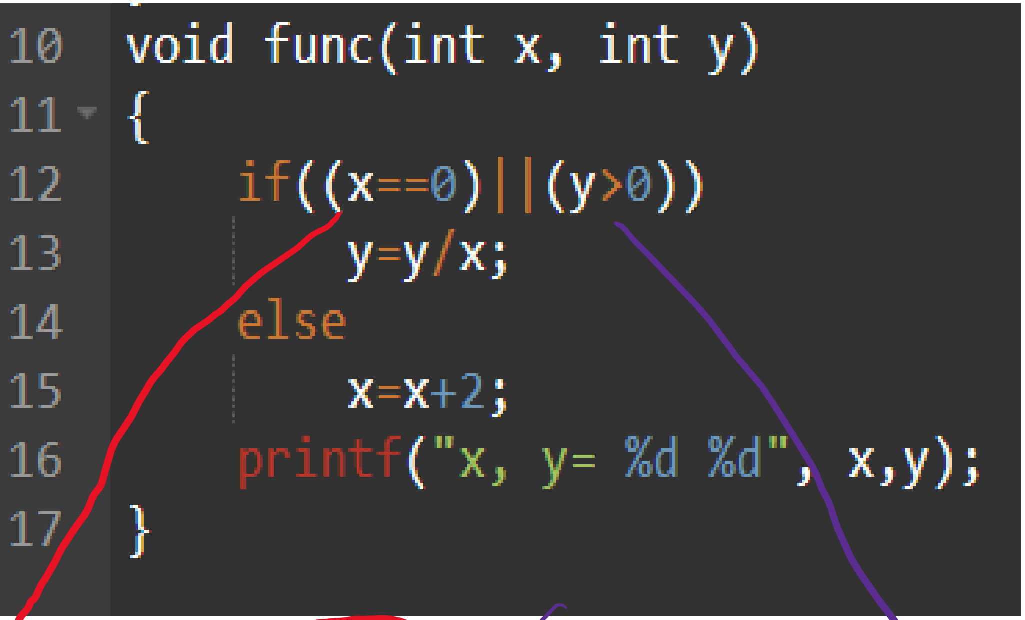
Try condition coverage.



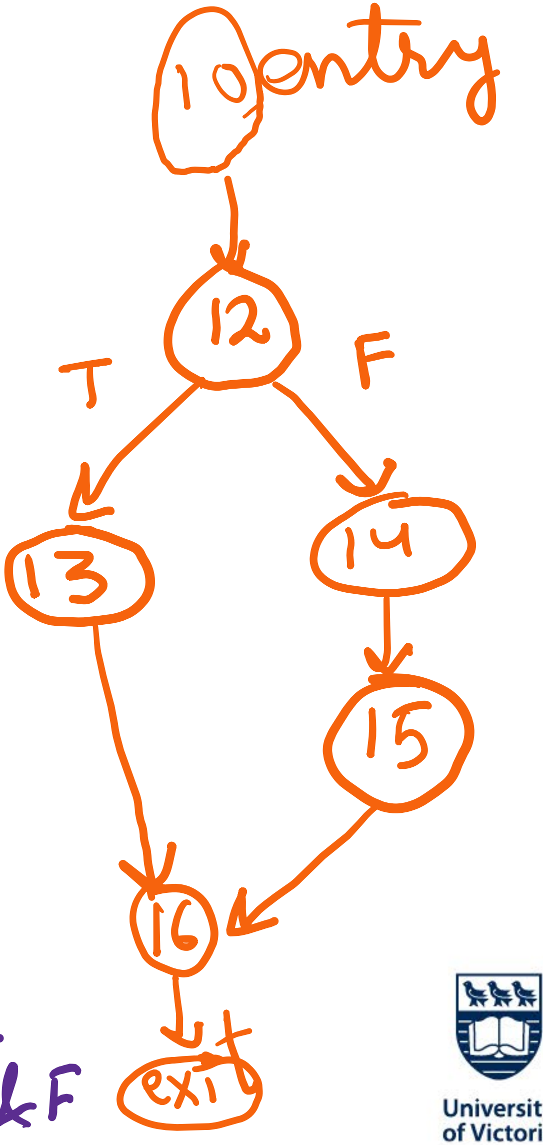
Branch coverage alone cannot reveal y/x error. It has to be combined with condition coverage to reveal all errors.

University of Victoria

Try to achieve 100% condition coverage



```
void func(int x, int y)
{
      if((x==0)||(y>0))
            y=y/x;
      else
            x=x+2;
      printf("x, y= %d %d", x,y);
}
```



Test 1: x=0, y=-5

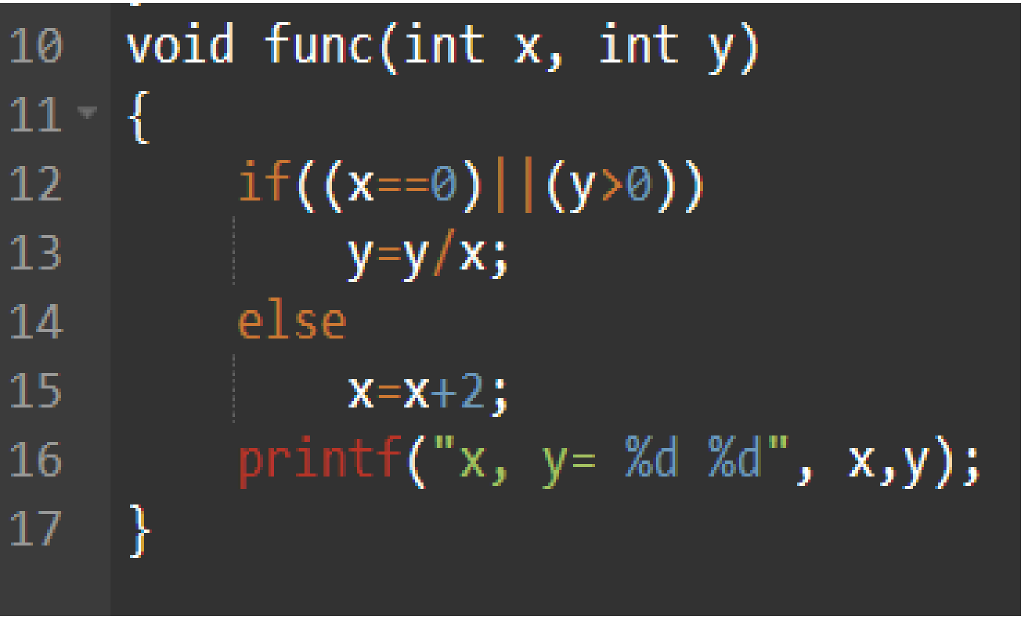
Test 2: x=5, y=5

100% condition coverage.

for condition (x==0) we have both T & F

For cond. (y>0) we have both T & F

What about branch coverage?



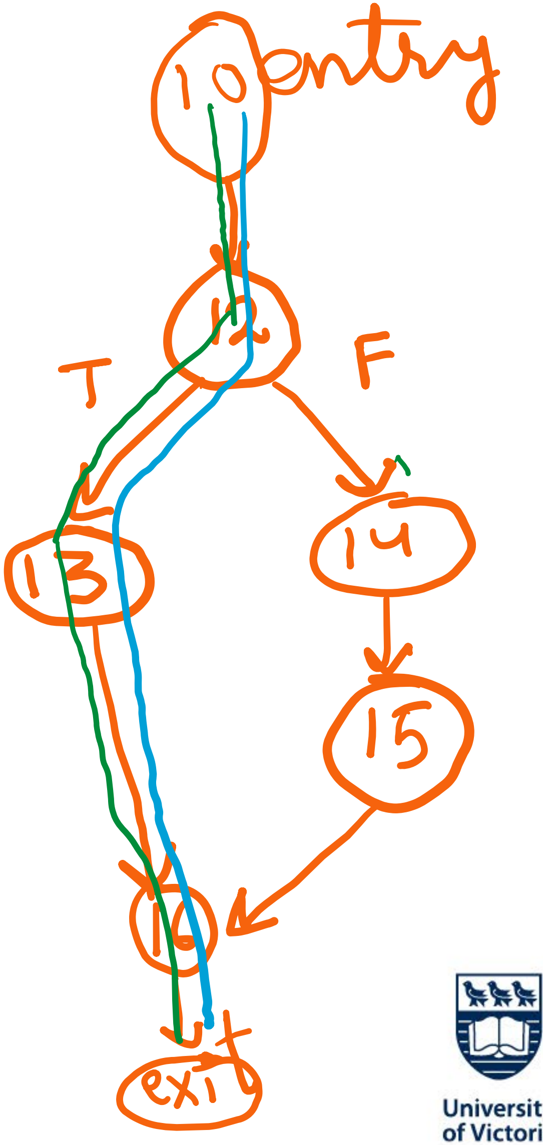
```
10  void func(int x, int y)
11  {
12      if((x==0)||(y>0))
13          y=y/x;
14      else
15          x=x+2;
16      printf("x, y= %d %d", x,y);
17  }
```

Test 1 : x=0 , y=-5

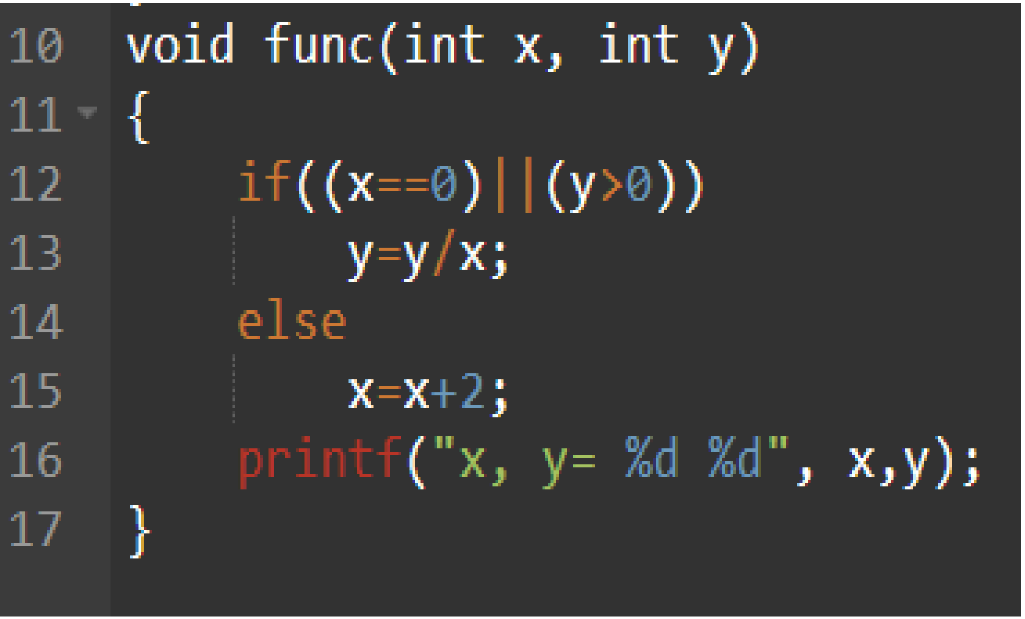
Test 2 : x=5, y=5

Branch coverage = 50%

even though condition coverage is = 100%



Add a test case to achieve a 100% Branch & Condition coverage



```
10  void func(int x, int y)
11  {
12        if((x==0)||(y>0))
13            y=y/x;
14        else
15            x=x+2;
16        printf("x, y= %d %d", x,y);
17  }
```

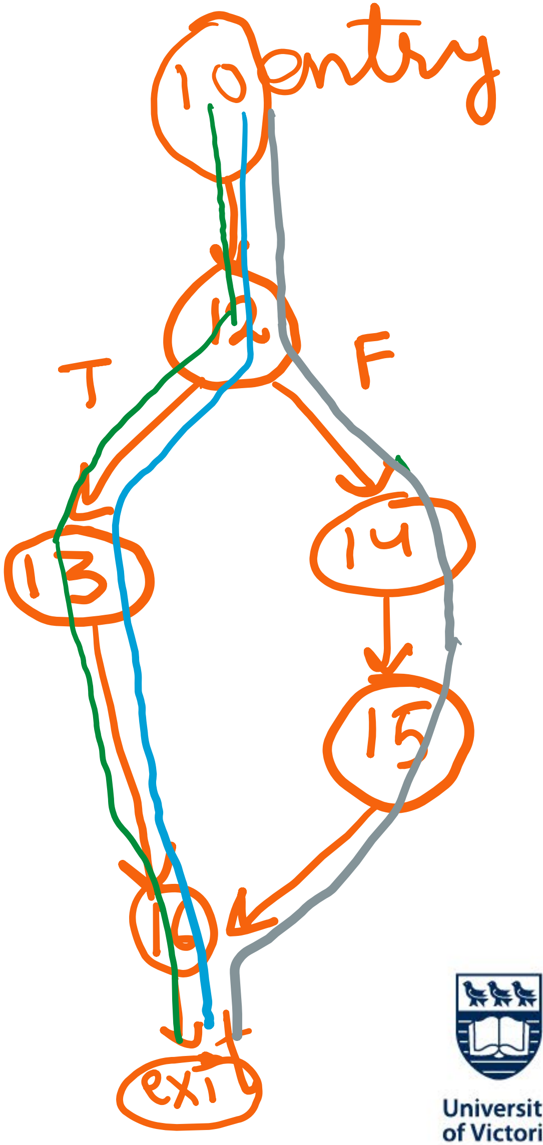
Test 1 : x=0, y=-5

Test 2 : x=5, y=5

TRUE branch execution

FALSE branch exec.

Test 3 : x=5, y=-5



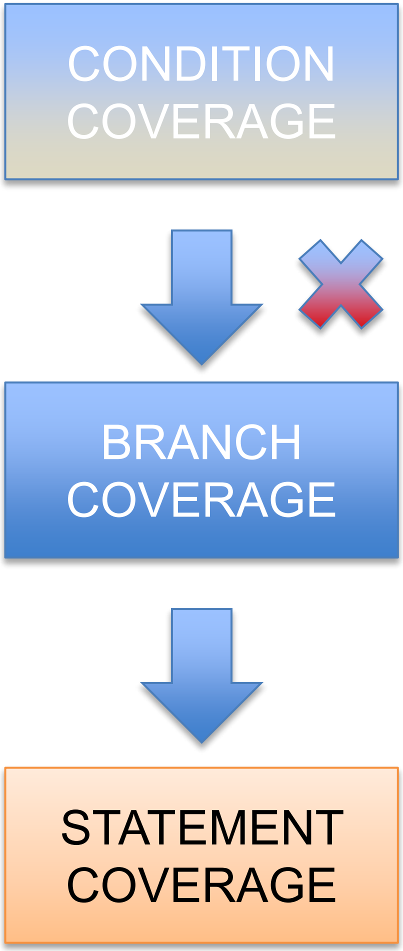
Condition + Decision coverage

- In practice, whenever we use condition coverage, we actually perform branch + condition coverage. In other words, we make sure that we achieve 100% condition coverage (i.e., all the outcomes of all conditions are exercised) and 100% branch coverage (all the outcomes of the compound decisions are exercised).

$$\text{C/DC coverage} = \frac{\text{conditions outcome covered} + \text{decisions outcome covered}}{\text{conditions outcome total} + \text{decisions outcome total}} \cdot 100$$

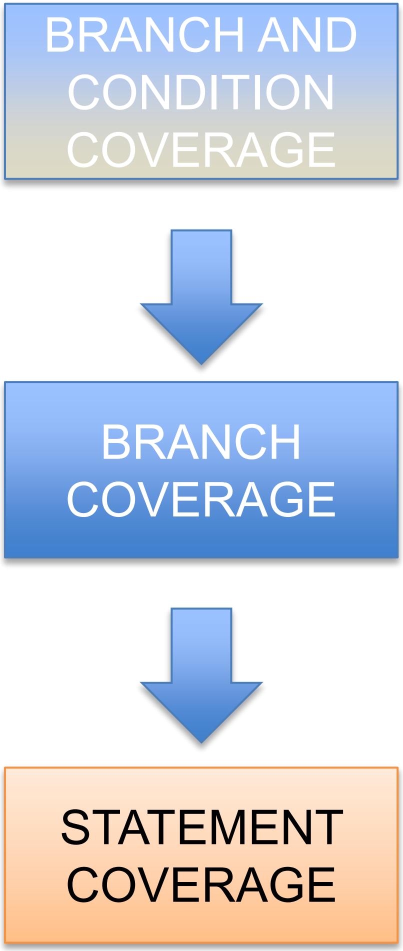
Question

- Does condition coverage imply branch coverage?
- NO

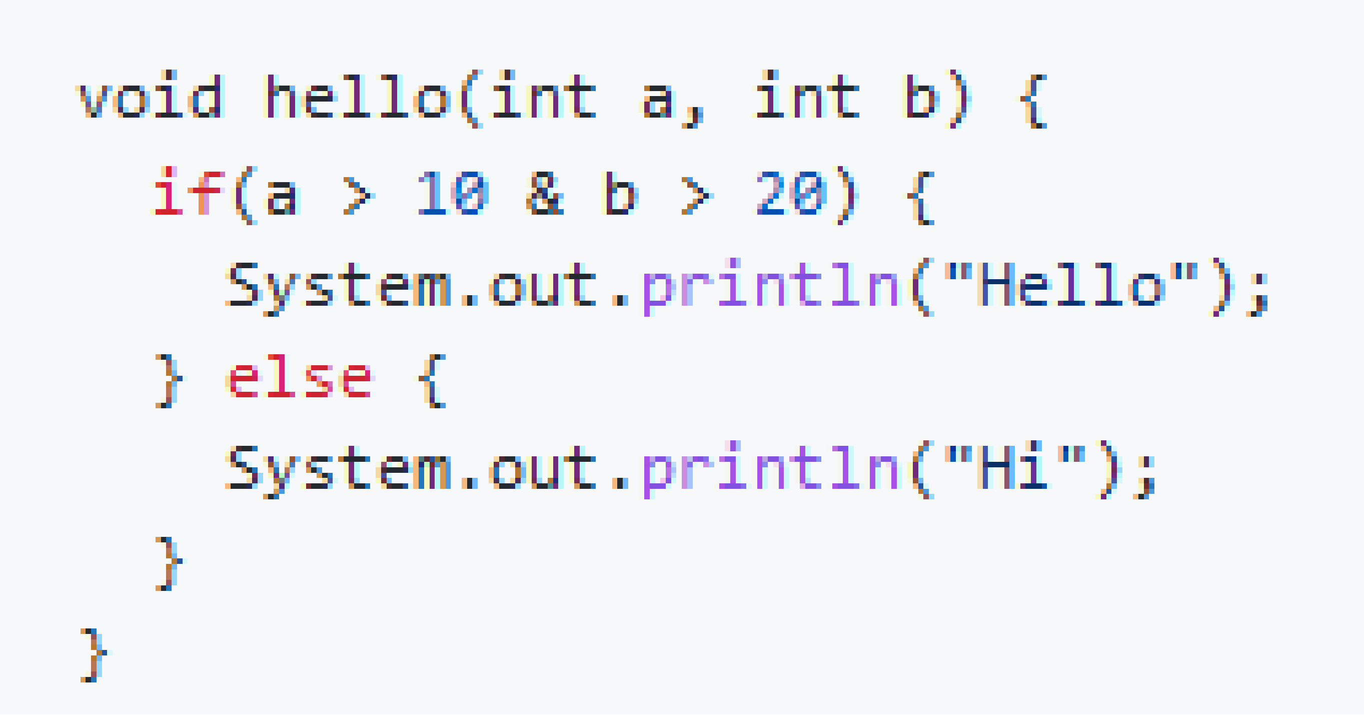


Question

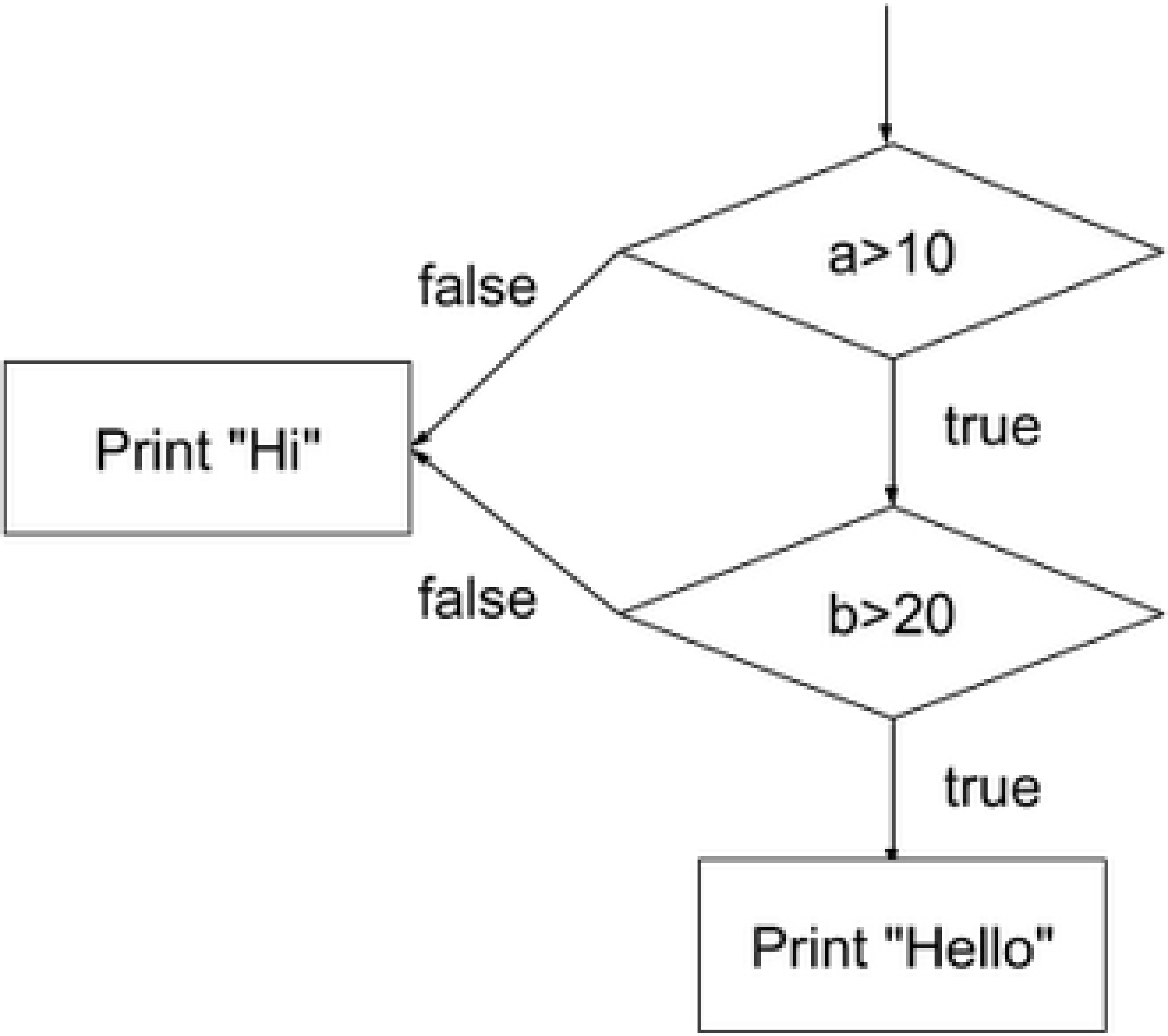
- Does branch and condition coverage imply branch coverage?
- YES
- So any test which satisfy branch and condition coverage would also satisfy branch coverage, condition coverage and statement coverage. So this is the strongest criteria.



Take another example :Look at the following program and its draw its CFG:



```java
void hello(int a, int b) {
  if(a > 10 & b > 20) {
    System.out.println("Hello");
  } else {
    System.out.println("Hi");
  }
}
```


a>10
false
true
Print "Hi"
b>20
false
true
Print "Hello"

Create test cases

- Create a test case which causes the first condition a > 10 to be true, and the second condition b > 20 to be false.
- Create a test case which causes the first condition false, and the second condition true.
- Are these two test cases sufficient?

Create test cases

- A test **T1 = (20, 10)** causes the first condition a > 10 to be true, and the second condition b > 20 to be false.
- A test **T2 = (5, 30)** makes the first condition false, and the second condition true. Note that T1 and T2 together achieve **100% basic condition coverage**. After all, **both conditions a and b have been exercised as both true and false.**

Create test cases

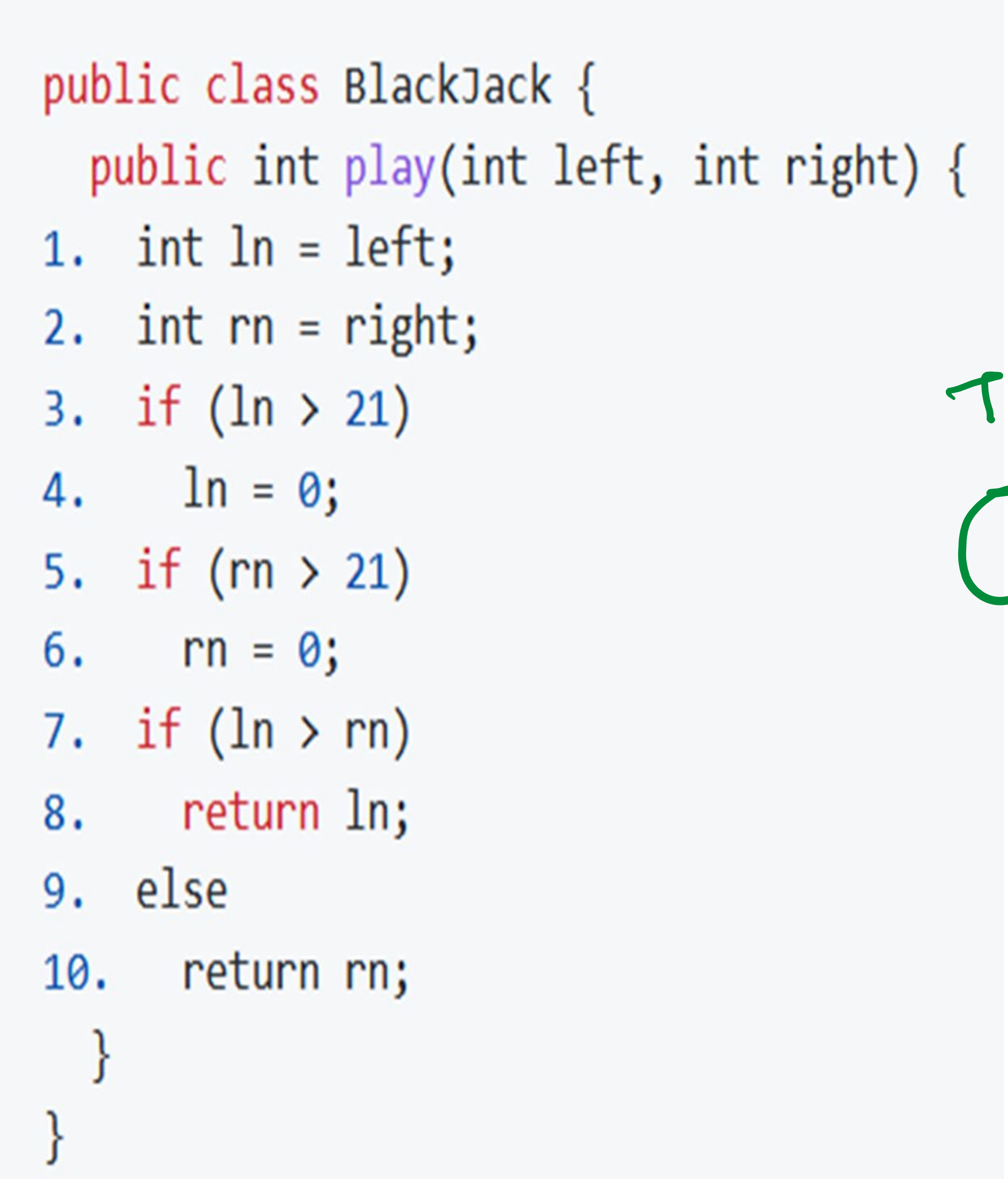
- However, the final outcome of the entire decision was false in both tests. We never saw this program printing "Hello".
- We found a case where we achieved 100% basic condition coverage, but only 50% branch coverage. This is not a smart testing strategy. This is why **looking only at the conditions themselves while ignoring the overall outcome of the decision block is called basic condition coverage.**
- Include another test case **T3 (30,30)** to execute ‘Hello’.

Create test cases

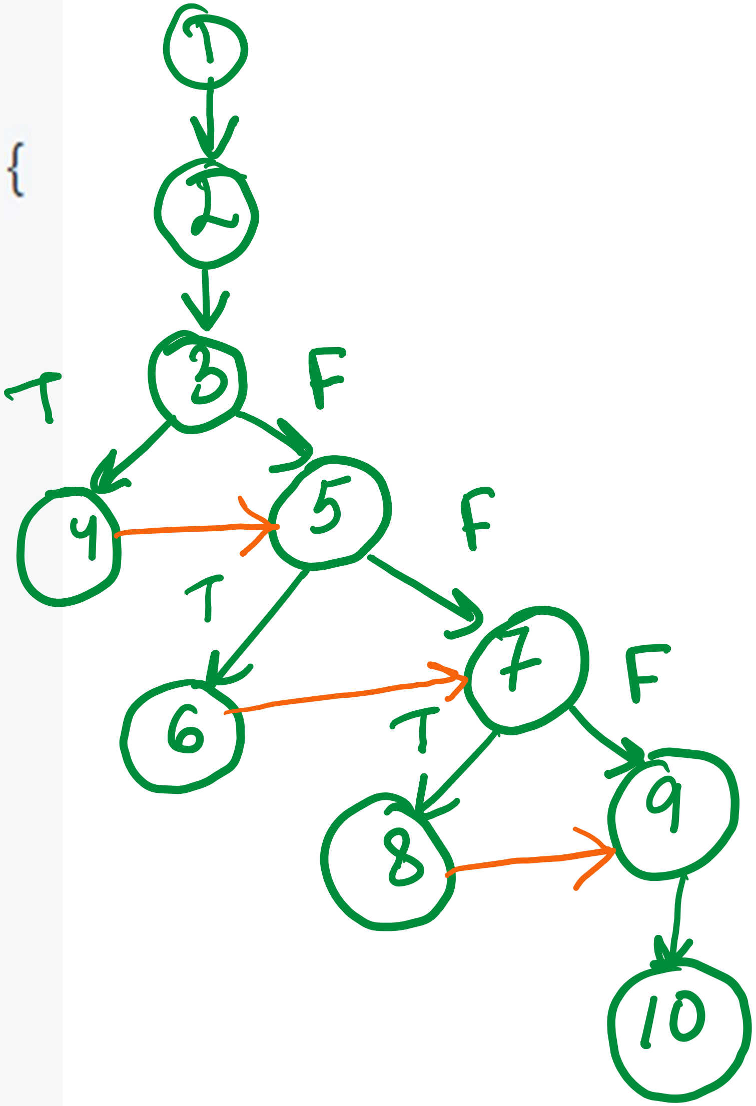
- However, the final outcome of the entire decision was false in both tests. We never saw this program printing "Hello".
- We found a case where we achieved 100% basic condition coverage, but only 50% branch coverage. This is not a smart testing strategy. This is why **looking only at the conditions themselves while ignoring the overall outcome of the decision block is called basic condition coverage.**
- Include another test case **T3 (30,30)** to execute ‘Hello’.

SOME PRACTICE CFG’S

University
of Victoria



```
public class BlackJack {
  public int play(int left, int right) {
1.  int ln = left;
2.  int rn = right;
3.  if (ln > 21)
4.    ln = 0;
5.  if (rn > 21)
6.    rn = 0;
7.  if (ln > rn)
8.    return ln;
9.  else
10.   return rn;
  }
}
```



Draw a CFG for this program

```
1. PrintDifference(int x, int y)
2. {
3.      while(x>y){
4.           if(x>0)
5.                x = x – y;
6.           else y = y – x;
7.                }
8.      print x, y ;
9. }
```