Multiple-Choice Questions:

1. True or false? Generally, in practice, developers exhaustively test software.

    a. True

    b. False

2. True or false? All "real" software contains bugs.

    a. True

    b. False

3. Which of the following is not a desirable quality of a unit test?

    a. No I/O

    b. Fast

    c. Non-deterministic

    d. Tests one property

    e. None of the above

4. Which of the following is true of *exhaustive testing*?

    a. Generally infeasible in practice

    b. Tests all possible inputs

    c. Typically results in an intractably large set of test cases even for small programs

    d. All of the above

    e. None of the above

5. Which of the following is <u>not</u> a difference between unit tests and integration tests?

    a. Unit tests should not perform I/O, whereas integration tests may do so

    b. Unit tests should be deterministic, whereas integration tests may have non-determinism

    c. Unit tests should be fast (less than half a second), whereas integration tests may be slower

    d. Unit tests must be black-box tests, whereas integration tests must be white-box tests

    e. None of the above (they are all differences)

6. Which of the following is <u>not</u> a difference between black-box and white-box testing?

    a. Black-box tests are based only on the interface of a component, whereas white-box tests are based on the implementation

    b. Black-box tests often focus on boundary cases, whereas white-box tests tend not to

    c. White-box tests often aim to achieve particular levels of code-coverage, whereas black-box tests do not

    d. In test-driven development, the developers generally write black-box tests, and not white-box tests

    e. None of the above (they are all differences)

7. In __________, you hook <u>everything</u> together and treat the system like a black box.

    a. test-driven development

    b. system testing

    c. unit testing

    d. integration testing

    e. None of the above

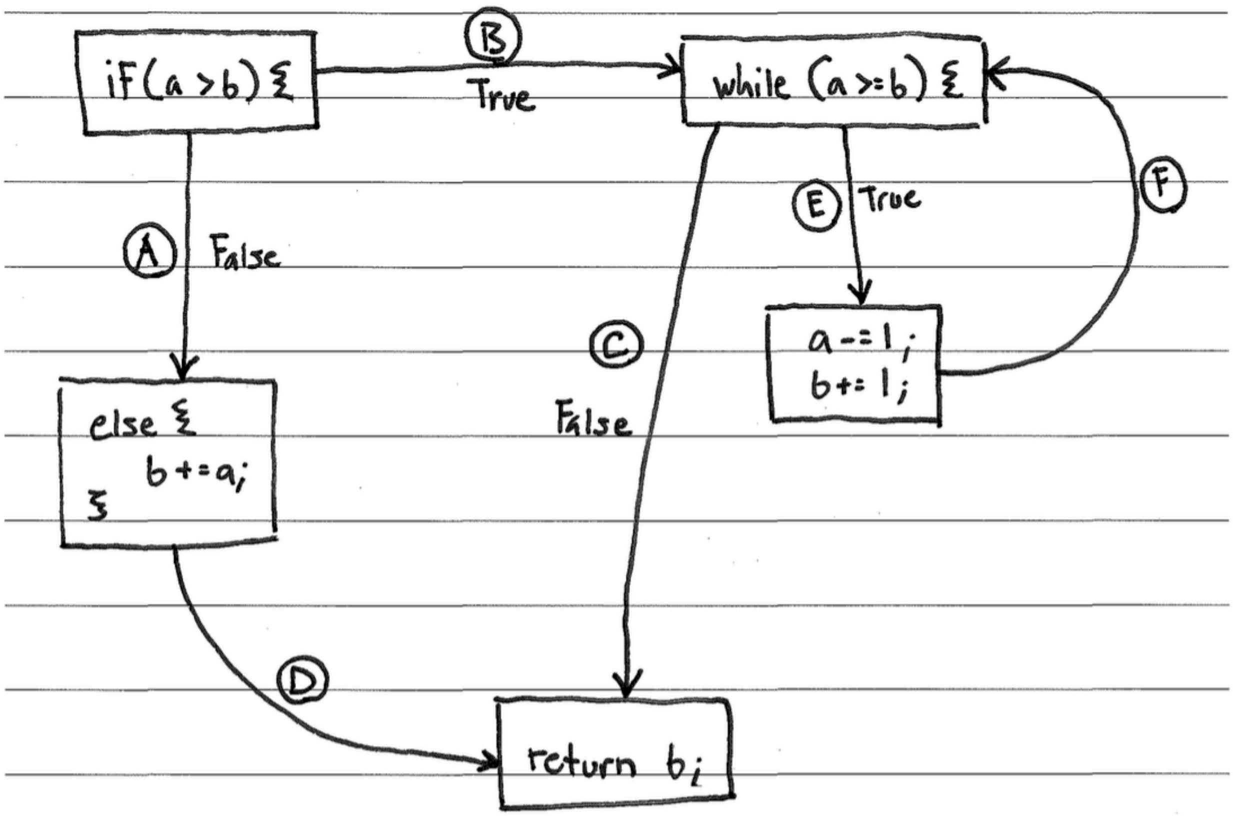
Solutions:

1. b
2. a
3. c
4. d
5. d
6. e
7. b

Problem: Draw a control flow diagram for this function. Label each edge with an uppercase letter.

```
int funWithNumbers(int a, int b) {
    if (a > b) {
        while (a >= b) {
            a -= 1;
            b += 1;
        }
    } else {
        b += a;
    }
    return b;
}
```

Solution:



if(a > b) {
B
True
while (a >= b) {
A
False
E
True
F
C
a -= 1;
b += 1;
False
else {
b += a;
}
D
return b;

Problem: Fill in the table below with a test suite that provides *path coverage* of the code from the previous question. Cover no more than 2 iterations of the loop. In the covers column, list the relevant labeled items in your CFG that each test case covers. If there is some part of the coverage that is impossible to cover, then list it in the covers column, and put "N/A" in the associated x and y cells. Some cells in the table may be left blank.

Input		Covers
x	y	

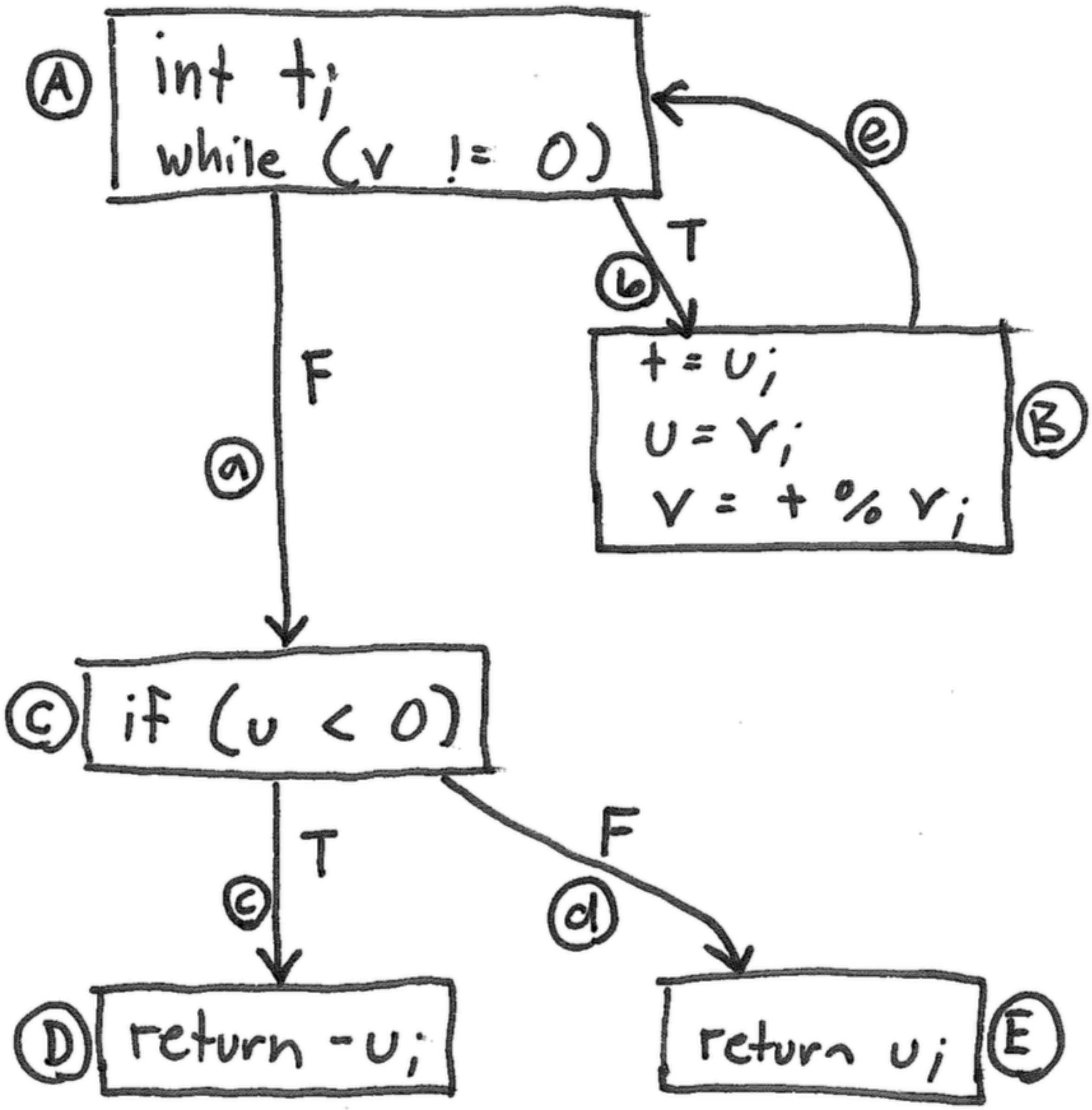
Solution:

Input		Covers
x	y	
1	2	AD
N/A	N/A	BC
1	0	BEFC
4	2	BEFEFC

Problem: Draw a control flow diagram for this function. Label each node in the graph with a capital letter, and label each edge with a lowercase letter.

```
int blammo(int u, int v) {
  int t;
  while (v != 0) {
    t = u;
    u = v;
    v = t % v; // Recall that % computes remainder of t/v
  }
  if (u < 0) { return -u; }
  return u;
}
```

Solution:



A
int t;
while (v != 0)
T
b
F
a
t = u;
u = v;
v = t % v;
B
e
C
if (u < 0)
T
c
F
d
D
return -u;
return u;
E

Problems:

1. Fill in the table below with a test suite that provides statement coverage of the “blammo” code. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers
u	v	

2. Fill in the table below with a test suite that provides path coverage of the “blammo” code. Cover no more than 1 iteration of the loop. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers
u	v	

Solutions:

1.

Input		Covers
u	v	
2	2	A, B, C, E
-1	0	A, C, D

2.

Input		Covers
u	v	
-1	0	a, c
0	0	a, d
-2	-2	b, e, a, c
2	2	b, e, a, d

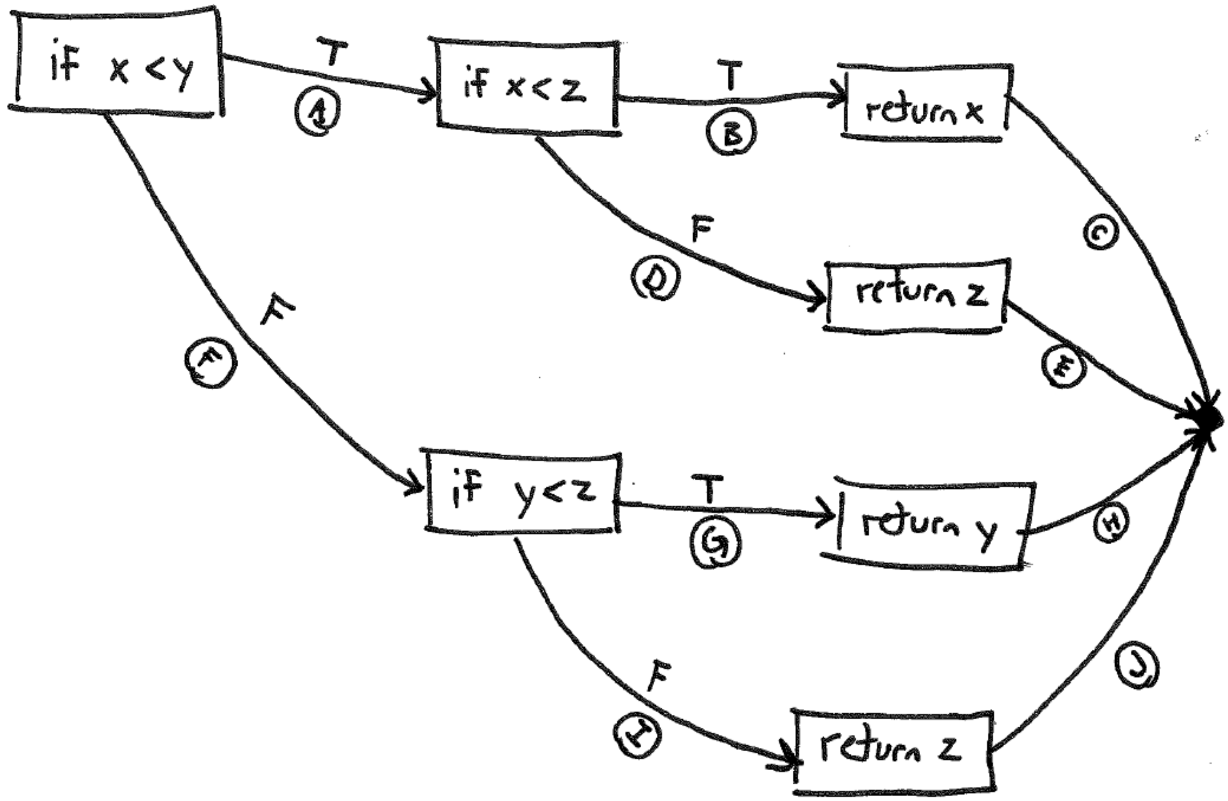
Paths:

a, c
a, d
b, e, a, c
b, e, a, d

Problem: Draw a control-flow graph for the following function. Label each edge in the graph with an uppercase letter.

```
def min_of_three(x, y, z)
  if x < y then
    if x < z then
      return x
    else
      return z
    end
  else
    if y < z then
      return y
    else
      return z
    end
  end
end
```

Solution:



if x<y
T
A
if x<z
T
B
return x
C
F
D
return z
E
F
F
if y<z
T
G
return y
H
F
I
return z
J

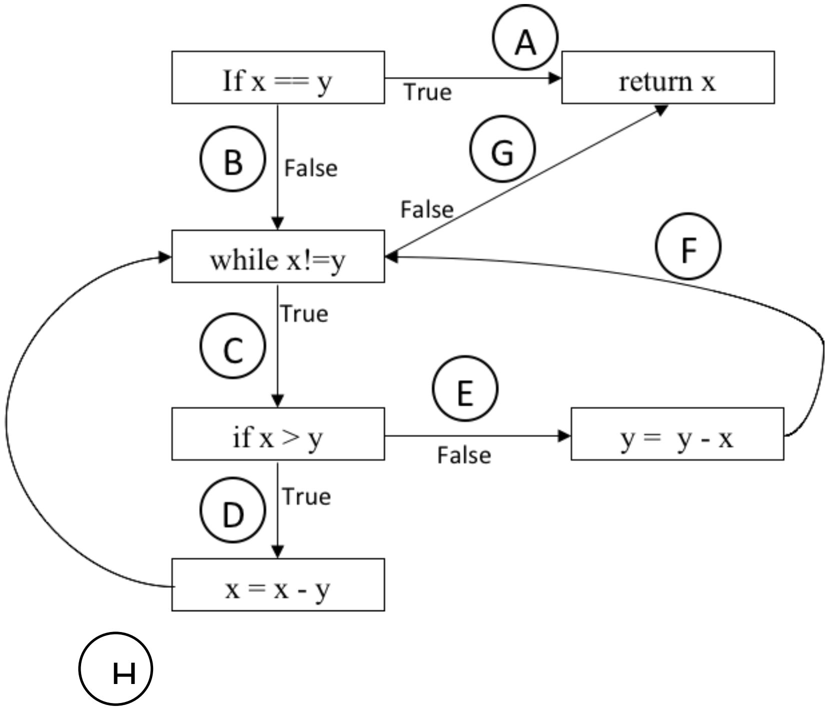
Problem: Fill in the table below with a test suite that provides <u>path coverage</u> of the `min_of_three` function from the previous question. In the covers column, list the relevant labeled edges in your CFG that each test case covers. Some cells in the table may be left blank.

Input			Expected Output	Covers
x	y	z		

Solution:

Input			Expected Output	Covers
x	y	z		
1	2	2	1	A, B, C
2	3	1	1	A, D, E
2	1	2	1	F, G, H
3	2	1	1	F, I, J

Consider the following control-flow graph for a `gcd` function in answering the questions below.



Problem: Fill in the table below with a test suite that provides condition coverage of the `gcd` function (see control-flow graph above). In the Covers column, list the relevant labeled edges in the CFG that each test case covers. Some cells in the table may be left blank.

Input		Expected Output	Covers
x	y		

Problem: Fill in the table below with a test suite that provides path coverage of the `gcd` function (see control-flow graph above). In the Covers column, list the relevant labeled edges in the CFG that each test case covers. Some cells in the table may be left blank. You need only cover executions that involve 1 iteration of the loop.

Input		Expected Output	Covers
x	y		

Solution: Condition Coverage

Input		Expected Output	Covers
x	y		
1	1	1	A
1	2	1	B, C, E, G
2	1	1	B, C, D, G
3	2		B, C, D, C, E, G

alternatively

Solution: Path Coverage

Input		Expected Output	Covers
x	y		
1	1	1	A
2	1	1	B, C, D, H, G
1	2	1	B, C, E, F, G
			B, G ← not possible