VU Programm- und Systemverifikation
Assignment 1: Assertions, Testing, and Coverage

Name: _____ Matr. number: _____

Due: April 22, 2021, 1pm

1 Coverage Metrics

Consider the following program fragment and test suite:

```
unsigned gcd (unsigned x, unsigned y) {
  unsigned m, k;
  if (x > y) {
    k = x;
    m = y;
  } else {
    k = y;
    m = x;
  }
  while (m != 0) {
    unsigned r = k % m;
    k = m;
    m = r;
  }
  return k;
}
```

Inputs		Outputs
x	y	return value
0	0	0
0	1	1
3	2	1

1.1 Control-Flow-Based Coverage Criteria (2.5 points)

Indicate (✓) which of the following coverage criteria are satisfied by the test-suite above.

	satisfied	
Criterion	yes	no
path coverage		✓
statement coverage	✓	
branch coverage	✓	
decision coverage	✓	

For each coverage criterion that is *not* satisfied, explain why this is the case:

In the presence of a loop, there are arbitrarily many paths; hence the path coverage criterion cannot be achieved.

1.2 Data-Flow-Based Coverage Criteria (4 points)

Indicate (✓) which of the following coverage criteria are satisfied by the test-suite above (here, the parameters of the function do not constitute definitions, the `return` statement is a c-use):

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses	✓	
all-c-uses/some-p-uses		✓
all-du-paths		✓

For each coverage criterion that is not satisfied, explain why this is the case:

- *The loop is only entered in the third test-case (in which* `x > y` *holds). Consequently the use* `k%m` *of the definitions* `k = y` *and* `m = x` *in the else-branch of the conditional statement is never reached, and the definitions* `k = y` *and* `m = x` *in the then-branch never reach the return statement.*
- *Since all-c-uses is not achieved, all-c-uses/some-p-uses cannot be achieved either.*
- *Since all-c-uses is not achieved, there are du-paths that have not been covered.*

1.3 Achieving Full Coverage (1 point)

Consider the two coverage criteria below.

- If the test-suite from above does not satisfy the coverage criterion, augment it with the *minimal* number of test-cases such that this criterion is satisfied. If full coverage cannot be achieved, explain why.
- If the coverage criterion is already achieved, explain why.

all-c-uses

Inputs		Outputs
x	y	result
2	3	1
1	0	1

all-uses

Inputs		Outputs
x	y	result
2	3	1
1	0	1

1.4 General Questions (2.5 points)

Indicate whether the following statements are true or false!

Statement	**True**	**False**
If a test-suite achieves `all-c-uses` and `all-defs` coverage, it also achieves `all-c-uses/some-p-uses` coverage.	☑	○
If a test-suite achieves path coverage, it also achieves statement coverage.	☑	○
If a test-suite does not achieve `all-uses` coverage, it cannot achieve `all-c-uses` either.	○	☑
For any given program, it is always possible to achieve MC/DC coverage.	○	☑
If full statement coverage cannot be achieved, then the program contains unreachable code.	☑	○

1.5 Formal Definition (1 point)

Let $S_{\rightarrow\bigcirc}, S_1, \ldots, S_n$ be the sets of reachable states at program locations $\rightarrow\bigcirc, 1, \ldots, n \in V$, respectively. Morover, given the conditional statement at program location ℓ, let C_ℓ be corresponding decision. Assume that all branches correspond to conditional statements.

Define an abstraction function α and an abstract domain such that

$$\forall i \in v \,.\, \alpha(S_i) = \top$$

if (and only if) full branch coverage can be reached.

$$\alpha(S_i) \stackrel{\text{def}}{=} \begin{cases} \{\sigma(B_i) \mid \sigma \in S_i\} & \text{if } i \text{ is branching statement with condition } B_i \\ \top & \text{otherwise} \end{cases}$$

(*We assume there are no unconditional branches.*)

2 Equivalence Partitioning and Boundary Testing

If you who had contact with a person who has tested positive for COVID-19, the function `classify` below will determine whether you had a category I contact, a category II contact, or neither.

```
category classify (int time,
                   int distance,
                   int their_age,
                   int your_age);
```

It uses the following data-types and parameters:

- `category` is an enum type defined as `enum priority {C1, C2, NEITHER};`
- `time` is the (cumulative) amount of time (in minutes) of exposure.
- `distance` is the minimal distance (in meters) to the infected person.
- `their_age` is the age of the person who tested postive for `COVID-19`.
- `your_age` is the age of the person that is to be classified.

The function `classify` is supposed to implement the following rules:

- A person who (cumulatively) spent at least 15 minutes in less than 2 meters distance from a person older than 10 years of age is classified as category I contact.
- Children younger than 10 years who were in contact (for at least 15 minutes in less than 2 meters distance) with a child also younger than 10 years that tested positive for COVID-19 are classified as category II contacts.
- A person who (cumulatively) spent less than 15 minutes with an infected person or kept a distance of at least 2 meters does not fall into category I or II.

Note: *The requirements are unclear about positively tested contact persons aged exactly 10. Therefore, we will have separate equivalence classes with* `age=10`.

2.1 Equivalence Partitioning (2 points)

From the specification above, derive equivalence classes for the function `classify`. Use the table below to partition them into *valid equivalence classes* (valid inputs) and *invalid equivalence classes* (invalid inputs). Label each of the equivalence classes clearly with a number (in the according column). For each correct *equivalence class* you can score $\frac{1}{2}$ a point (up to 2 points).
(Do not provide test-cases here – that's task 2.2)

2.1.1 Valid Equivalence Classes

Condition	**ID**
$(\texttt{time} \geq 15) \wedge (\texttt{distance} < 2) \wedge (\texttt{their\_age} > 10)$	1
$(\texttt{time} \geq 15) \wedge (\texttt{distance} < 2) \wedge (\texttt{your\_age} < 10) \wedge (\texttt{their\_age} < 10)$	2
$(\texttt{time} \geq 15) \wedge (\texttt{distance} < 2) \wedge (\texttt{your\_age} < 10) \wedge (\texttt{their\_age} = 10)$	3
$(\texttt{time} < 15) \vee (\texttt{distance} \geq 2)$	4

Note that we didn't choose to create separate equivalence classes for $(\texttt{time} \geq 15)$, $(\texttt{distance} < 2)$, *. . . ; the reason is, that the outcome depends on a combination of the parameters.*

2.1.2 Invalid Equivalence Classes

Condition	**ID**
$(\texttt{time} < 0)$	5
$(\texttt{distance} < 0)$	6
$(\texttt{your\_age} < 0)$	7
$(\texttt{their\_age} < 0)$	8

2.2 Boundary Value Testing (2 points)

Use *Boundary Value Testing* to derive a test-suite for the function `triage`. Specify the inputs points for `classify`. Indicate clearly which equivalence classes each test-case covers by referring to the numbers from task (a). You can receive up to 2 points ($\frac{1}{2}$ a point per test-case), where redundant test-cases and test-cases that do not represent boundary values do not count.

Input	**Output**	**Classes Covered**
$(15, 1, 18, 11)$	`C1`	1
$(15, 0, 0, 11)$	`C1`	1
$(15, 1, 9, 9)$	`C2`	2
$(15, 0, 9, 0)$	`C2`	2
$(15, 1, 9, 10)$	`C2` (?)	3
$(14, 0, 18, 42)$	`NEITHER`	4
$(15, 2, 18, 42)$	`NEITHER`	4
$(-1, 0, 18, 42)$	error	5
$(15, -1, 18, 42)$	error	6
$(15, 2, -1, 42)$	error	7
$(15, 2, 18, -1)$	error	7
...		

3 Invariants (5 points)

Consider the following program, where `t`, `x`, `y`, and `z` are integer values in $\mathbb{Z}$ (that means no over- or underflow can happen):

```
z = 1;
if (x > y) {
  int t = x; x = y; y = t;
}
while (y > x) {
  x = x + 1;
  y = y - 1;
  z = z + (y - x);
}
```

Consider the formulas below; tick the correct box (☑) to indicate whether they are loop invariants for the program above.

- If the formula is an inductive invariant for the loop, provide an informal argument that the invariant is inductive.
- If the formula P is an invariant that is *not* inductive, give values of `x`, `y`, and `z` before and after the loop body demonstrating that the Hoare triple

  $$\{P \wedge B\} \quad \texttt{x} = \texttt{x} + 1;\ \texttt{y} = \texttt{y} - 1;\ \texttt{z} = \texttt{z} + (\texttt{y} - \texttt{x}) \quad \{P\}$$

  (where B is `(y > x)`) does not hold. This means that you need to find a starting state in which $P \wedge B$ evaluates to true and which results in a violation of P after executing the loop body.
- Otherwise, provide values of `x`, `y`, and `z` that correspond to a reachable state showing that the formula is *not* an invariant.

$(x \leq y)$	☐ Inductive Invariant	☐ Non-inductive Inv.	☑ Neither

Justification:

Before: x = 0, y = 1, z arbitrary
After: x = 1, y = 0

$(x - y) \leq 1$	☑ Inductive Invariant	☐ Non-inductive Inv.	☐ Neither

Justification:
Base: $x \leq y$ implies $(x - y) \leq 1$
Induction: Pre-condition of $x - y \leq 1$ is $x - y < 0$, implied by $y > x$

$(y - x) + 2 \neq 0$	☑ Inductive Invariant	☐ Non-inductive Inv.	☐ Neither

Justification:
Note: This invariant is also implied by $(x - y) \leq 1$.
Base: $x \leq y$ implies that $y - x$ is positive, hence adding 2 can't result in 0
Induction: if $y > x$, then $y - x$ is at least one, and after the loop at least -1.
Hence adding 2 can't make the term 0.

$z \geq 0$	☐ Inductive Invariant	☑ Non-inductive Inv.	☐ Neither

Justification:
Before: $z = 0, y = 1, x = 0$
After: $x = 1, y = 0, z = 0 + (0 - 1)$
We know that $-1 \leq (y - x)$ when z is assigned, and $(y - x) = -1$ only if $y = x + 1$ upon loop entry. So after z is decreased once by 1, the loop terminates.
Note that z is initially 1.

$z \geq 1$	Inductive Invariant	☐ Non-inductive Inv.	☑ Neither

Justification:
Start with $z = 1, x = 0, y = 1$, the $z = 0$ after the loop.