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

Name: ________________________________ Matr. number: _______________

Due: April 30, 1pm

1 Coverage Metrics

Consider the following program fragment and test suite:

```
unsigned gcd (unsigned m, unsigned n) {
  unsigned i;
  if (m > n) {
    i = n;
  } else {
    i = m;
  }
  bool done = false;
  while ((i > 0) && !done) {
    if ((m % i == 0) && (n % i == 0) {
        done = true;
    } else {
        i = i - 1;
    }
  }
  return i;
}
```

Inputs		Outputs
m	n	return value
0	1	0
1	0	0
1	1	1
2	3	1

1.1 Control-Flow-Based Coverage Criteria (3 points)

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

	satisfied	
Criterion	yes	no
statement coverage		
decision coverage		
condition coverage		
modified condition/decision coverage		

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

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-p-uses/some-c-uses		

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

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.

MC/DC

Inputs		Outputs
`m`	`n`	result

all-p-uses

Inputs		Outputs
`m`	`n`	result

1.4 Modified Condition/Decision Coverage (1 point)

Consider the expression $((a \vee b) \wedge c)$, where a, b, and c are Boolean variables. Provide a *minimal* number of test cases such that modified condition/decision coverage is achieved for the expression. Clarify <u>for each test case</u> *which* condition(s)/value(s) independently affect(s) the outcome.

MC/DC

Inputs			Outcome
`a`	`b`	`c`	`(a || b) && c`

2 Equivalence Partitioning and Boundary Testing

The resources for performing RT-PCR tests to determine whether a patient has contracted the COVID-19 virus are extremely limited; consequently, critical patients will have to be prioritized. The function

```
priority triage (enum countries travel,
                 enum symptoms sympt,
                 int age);
```

is used to determine the priority with which a person should be tested or not. It uses the following data-types:

- `priority` is an enum type defined as `enum priority {high, medium, low};`
- `countries` is an enum type listing 196 countries used to represent the country the patient most recently traveled to (if any). It is defined as follows:

  ```
  enum countries { None = 0, China = 1, Iran = 2, Italy = 3, ...};
  ```

  The first entry (0) indicates that the patient has not traveled outside Austria recently; the following k entries are countries that are classified as critical, and the remaining $196 - k$ entries are countries that are (still) considered safe.
- `symptoms` is an enum type listing 100 symptoms defined as follows:

  ```
  enum countries {
    None = 0, Tiredness, Aches, Cough, Fever, ..., };
  ```

  The first entry (0) indicates that the patient has no symptoms, the following m symptoms are common symptoms of COVID-19, and the remaining symptoms ($m + 1$ to 100) are not known to be related to the new virus.
- The parameter `age` represents the age of the patient.

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

- Patients who have no recent travel history to critical countries or show none of the typical symptoms are considered low priority.
- Patients who have traveled to a country classified as critical and report a relevant symptom are medium priority if they are younger than 65, and high priority if they are 65 and above.

2.1 Equivalence Partitioning (3.5 points)

From the specification above, derive equivalence classes for the function `triage`. 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 3.5 points).
(Do not provide test-cases here – that's task 2.2)

2.1.1 Valid Equivalence Classes

Condition	ID

2.1.2 Invalid Equivalence Classes

Condition	ID

2.2 Boundary Value Testing (3.5 points)

Use *Boundary Value Testing* to derive a test-suite for the function `triage`. Specify the inputs points for `triage`. Indicate clearly which equivalence classes each test-case covers by referring to the numbers from task (a). You can receive up to 3.5 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**

3 Invariants (4 points)

Consider the following program, where `x` and `y` are non-negative natural numbers (possibly 0):

```
x = y + 1;
while (x != y) {
  x = x + (y % 2);
  y = y + (x % 2);
}
```

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` and `y` before and after the loop body such that $P \wedge B$ (where B is `(x != y)`) holds before the execution of

  $$\mathtt{x} = \mathtt{x} + (\mathtt{y}\%2);\ \mathtt{y} = \mathtt{y} + (\mathtt{x}\%2);$$

  and P does not hold anymore afterwards.
- Otherwise, provide values of `x` and `y` that correspond to a reachable state showing that the formula is *not* an invariant.

$(x - y) \leq 1$ ☐ Inductive Invariant ☐ Non-inductive Inv. ☐ Neither   Justification:
$(x - y) \leq 2$ ☐ Inductive Invariant ☐ Non-inductive Inv. ☐ Neither   Justification:
$(x - y)\%2 = 1$ ☐ Inductive Invariant ☐ Non-inductive Inv. ☐ Neither   Justification:
$(x \geq y)$ ☐ Inductive Invariant ☐ Non-inductive Inv. ☐ Neither   Justification: