SE WS23 (194.020) – **Test 2, Gruppe A**, Dauer: 75min 21.12.2023

Nachname: ______________________ Matrikelnummer: [ ][ ][ ][ ][ ][ ][ ][ ]

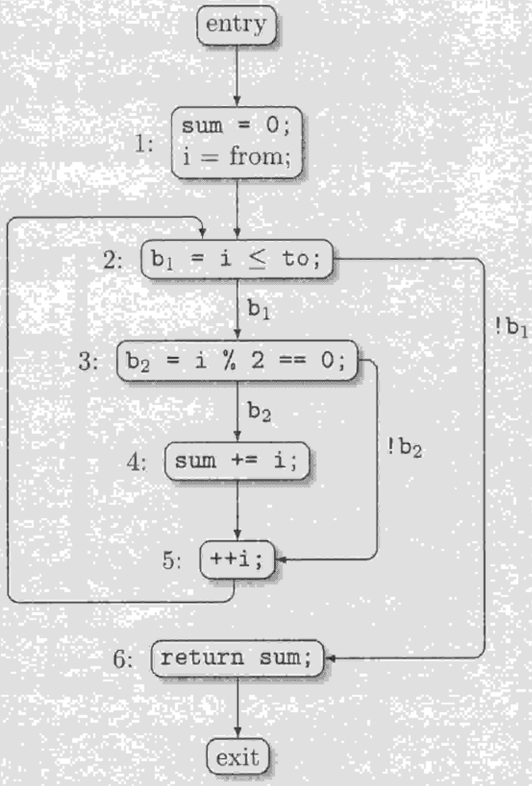
Vorname: ______________________

- *For questions that ask you to design, "minimum number of test cases", 1 point is deducted per redundant test.*

1.) Control-Flow Coverage (24 Points)

Given two integers `from` and `to`, function `sumEven` sums the even numbers in range [*from*, *to*].

```java
public int sumEven(int from, int to) {
  int sum = 0;
  for (int i = from; i <= to; ++i) {
    if (i % 2 == 0) {
      sum += i;
    }
  }
  return sum;
}
```



1. How many basic blocks and branches are there in the given CFG? Do not count the entry and exit blocks.

   Basic blocks: ____ Branches: ____

2. How many basic blocks and branches does the test case `(from=1, to=1)` cover? Compute basic-block coverage and branch coverage (as fractions).

   Basic-block coverage: ____/____ Branch coverage: ____/____

3. Design a test case that reaches 100% basic-block coverage, but not 100% branch coverage.

   `from=`______ `to=`______

4. Design a test case that reaches 100% branch coverage.

   `from=`______ `to=`______

5. Design test cases that reach 100% loop coverage.

   Hint: Use as many lines as you need!

   `from=`______ `to=`______
   `from=`______ `to=`______
   `from=`______ `to=`______
   `from=`______ `to=`______

2.) Design By Contract (32 Points)

Consider the following two function signatures for functions `f` and `g`.

```
int f(int x);
int g(int x);
```

The pre- and postcondition for function `f` are

- Precondition: input value $x \in [0, 10000]$
- Postcondition: return value $y \in [0, 10000]$

The pre- and postcondition for function `g` are

- Precondition: input value $x \in \{2, 3, 5, 7, 11, 13\}$
- Postcondition: return value $y \in \{2, 3, 5, 7\}$

For each implementation of `f`, mark only the appropriate cells:

2 points for each correct 3-cell block, 0 points for a 3-cell block containing at least one error.

Implementation	The precondition ...			The postcondition ...		
	is weakened	is strengthened	breaks the expected behavior	is weakened	is strengthened	breaks the expected behavior
input value $x \in [0, 0]$ return value $y \in [0, 10001]$						
input value $x \in [-12345, 12345]$ return value $y \in [0, 999]$						
input value $x \in [0, 100]$ return value $y \in [100, 100]$						
input value $x \in [0, \infty]$ return value $y \in [0, \infty]$						

For each implementation of `g`, mark only the appropriate fields:

Implementation	The precondition ...			The postcondition ...		
	is weakened	is strengthened	breaks the expected behavior	is weakened	is strengthened	breaks the expected behavior
input value $x \in \{1, 2, 3, 5, 7, 11, 13\}$ return value $y \in \{2, 3, 5, 7, 11, 13\}$						
input value $x \in \{2, 3, 5, 7, 11, 13, 14\}$ return value $y \in \{11, 13\}$						
input value $x \in \{2, 3, 5\}$ return value $y \in \{2, 3, 5\}$						
input value $x \in [-\infty, \infty]$ return value $y \in \{2, 3, 5\}$						

3.) Theory Questions (22 Points)

Choose the correct answer for each of the following questions.
correct answer ⇒ 2 points, incorrect answer ⇒ -2 points, no answer ⇒ 0 points, minimum 0 points for this task

- Ensuring that the system implements requirements correctly is considered to be ...

  ☐ verification. ☐ validation.

- Testing all possible inputs of a system component is ...

  ☐ always ☐ typically not ... possible.

- Writing test cases using program requirements is considered to be ...

  ☐ structural ☐ property-based ☐ specification-based ☐ exhaustive ... testing.

- Writing test cases based on code coverage is considered to be ...

  ☐ structural ☐ property-based ☐ specification-based ☐ exhaustive ... testing.

- A test double that returns hard coded answers is considered to be a ...

  ☐ dummy object. ☐ fake object. ☐ stub. ☐ mock. ☐ spy.

- A test double that wraps around an implementation and records its actions is considered to be a ...

  ☐ dummy object. ☐ fake object. ☐ stub. ☐ mock. ☐ spy.

- Given is a list of found software bugs from an e-commerce company. For each bug listed, find the best strategy from the testing pyramid:

  - Bug #101: Email service component is unable to retrieve certain customer emails from database component

    ☐ unit testing ☐ integration testing ☐ system testing ☐ manual testing

  - Bug #102: Address validation function does not accept addresses from Germany

    ☐ unit testing ☐ integration testing ☐ system testing ☐ manual testing

  - Bug #103: Function `purchase` calculates the price incorrectly

    ☐ unit testing ☐ integration testing ☐ system testing ☐ manual testing

- An object graph is used to show that an Alloy predicate is ...

  ☐ consistent. ☐ inconsistent.

- An object graph is used to show that an Alloy assertion is ...

  ☐ valid. ☐ invalid.

4.) Specification-Based Testing (14 Points)

1. Consider a large online shop with thousands of customers every day. On a normal day, the company considers a visitor count over 10.000 a success. On days with a sale announcement, the visitor count has to be over 20.000 to be considered a success. Function `isSuccess1` implements this functionality. Occasionally, the visitor tracking system is in maintenance mode, yielding negative numbers. For negative numbers, function `isSuccess1` throws an exception.

```
public boolean isSuccess1(int visitorCount , boolean isSale);
```

   - What are the partitions for each parameter?

   - How many partitions are there in total (after combining the above partitions without merging any)?
     Partitions: ____

2. After reviewing the specification, the company comes to the conclusion that distinguishing days based on sale announcements is not a good idea. They decided to drop the `isSale` parameter and consider every day as a normal day (i.e. no sale announcements). Function `isSuccess2` implements this functionality and behaves identically to `isSuccess1(visitorCount, false)`.

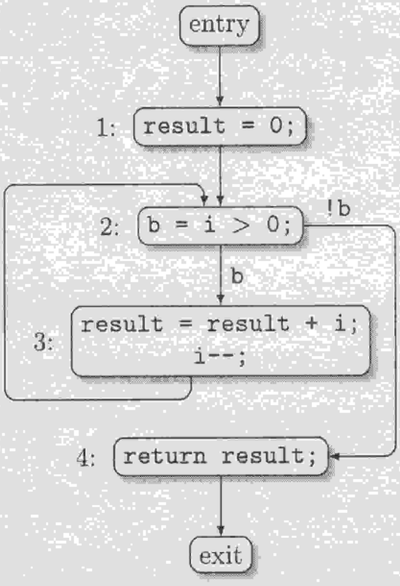
```
public boolean isSuccess2(int visitorCount);
```

   - How many partitions are there for parameter `visitorCount` of function `isSuccess2`?
     Partitions: ____
   - Design the minimum number of test cases for function `isSuccess2` according to a boundary value analysis.
     Hint: Use as many lines as you need!

     visitorCount=______
     visitorCount=______
     visitorCount=______
     visitorCount=______
     visitorCount=______
     visitorCount=______

5.) Data-Flow Coverage (28 Points)

```
public int sum(int i) {
  int result = 0;
  while (i > 0) {
    result = result + i;
    i--;
  }
  return result;
}
```



1. Apply the algorithm for computing *reaching definitions* for variable `result`, where *n* is the block number in the control-flow graph.

n	Reach(n)	ReachOut(n)
1		
2		
3		
4		

2. List the DU pairs for variable `result`.

   Hint: Use as many lines as you need!

Definition Block	Use Block

3. Instrument the code as shown in the lecture to measure DU-pairs coverage. What is the state of maps `defCover` and `useCover` after running the test case (`i=1`)? You may assume the maps start freshly initialized.

```
defCover['result'] = ______
useCover['result', ______, ______] = ______
useCover['result', ______, ______] = ______
useCover['result', ______, ______] = ______
useCover['result', ______, ______] = ______
useCover['result', ______, ______] = ______
useCover['result', ______, ______] = ______
```

4. Design the minimum number of test cases that reach 100% DU-pairs coverage for variable `result`.

   Hint: Use as many lines as you need!

```
i=______
i=______
i=______
i=______
```

6.) MC/DC (30 Points)

```java
public int compute(int a, int b, int c) {
  if ((a > b && a > c) || c == 1) {
    return 0;
  } else {
    return 1;
  }
}
```

1. How many branches and condition values does the function have?

   Branches: ____ Condition values: ____

2. How many branches and condition values does the test case (a=1, b=2, c=2) cover? Compute C+B coverage (as a fraction).

   Condition values covered: ____ Branches covered: ____ C+B coverage: ____/____

3. Design the minimum number of test cases that reach 100% C+B coverage. List for each test case which conditions are true and which are false. For each test case include the final value of the entire `if`-decision.

   Hint: Use as many lines as you need!

Inputs			Conditions			Decision
a	b	c	a>b	a>c	c==1	(a>b && a>c) || c==1

4. Design the minimum number of test cases that reach 100% MC/DC. List for each test case which conditions are true and which are false. For each test case include the final value of the entire `if`-decision.

   Hint: Use as many lines as you need!

Test id	Inputs			Conditions			Decision
	a	b	c	a>b	a>c	c==1	(a>b && a>c) || c==1
T1							
T2							
T3							
T4							
T5							
T6							

Give the independence pair for each condition using the test ids.

a>b : ____ – ____

a>c : ____ – ____

c==1: ____ – ____