Exercise 1: Equivalence Classes & Boundary Values

A local flower shop offers discounts to its most valuable customers. Function `discountPercent` calculates the percentage of discount that should be applied to a purchase. Purchases of more than 10 flowers are given 10% discount. Customers with membership cards are given 5% discount. (The discounts are cumulative.) Function `discountPercent` throws an `InvalidOrderException` if the number of flowers is zero.

```
1 public int discountPercent(int flowers, boolean membershipCard);
```

a) What are the partitions for each parameter? How many partitions are there in total?

```
flowers (int):
 - <0: UB (Assumption: InvalidOrderException)
 - =0: InvalidOrderException
 - 1-10: discountPercent += 0
 - >10: discountPercent += 10

membershipCard (bool):
 - false: discountPercent += 0
 - true: discountPercent += 5

Partition count: 6
```

b) What partitions can be combined?

```
 - flowers <= 0; membershipCard does not matter; InvalidOrderException
 - 1 <= flowers <= 10; membershipCard==false => return 0
 - 1 <= flowers <= 10; membershipCard==true => return 5
 - flowers > 10; membershipCard==false => return 10
 - flowers > 10; membershipCard==true => return 15
```

c) What are the boundary values? Which are the on and off points?

```
flowers (int):
 - 0
 - 10

membershipCard (bool):
 - true
 - false
```

d) Construct test cases for function `discountPercent` according to your analysis. Give inputs and expected outputs.

Input		Output
flowers	membershipCard	
-1	false	InvalidOrderException
0	true	InvalidOrderException
1	false	0
5	true	5
10	false	0
11	true	15
12	false	10

Exercise 2: Specification-Based & Structural Testing

a) Which of these software testing activities correspond to specification-based testing, which correspond to structural testing, and which correspond to neither?

```
Structural testing uses the structure of the source code to guide testing
Specification-based testing uses the program requirements as testing input
```

(a) Asking a colleague to check if the tests match the documentation.

```
Specification-based
```

(b) Measuring which statements are executed by each test case.

```
Structural
```

(c) Doing test-driven development.

```
Neither
```

(d) Testing with random data to find crashes.

```
Neither
```

(e) Constructing test cases to cover all branches.

```
Structural
```

b) Which of the following statements are correct?

(a) MC/DC is a stronger property than branch coverage.

```
True
```

(b) Programs that have 100% path coverage do not contain any kind of bugs.

```
False, no testing method can guarantee the program is bug free
```

(c) Boundary values are extracted from the source code.

```
False, they are based on the inputs and their domain
```

(d) Loop coverage is a stronger property than branch coverage.

```
False, loop coverage is in fact often combined with branch coverage
```

(e) A test suite constructed from boundary values has 100% branch coverage.

```
False, since we "pragmatically decide" which partitions to combine we could
miss a branch
```

Exercise 3: Basic-Block & Branch Coverage

```
1  public int compute(int[] x) {
2    if (x == null) {
3      return 0;
4    }
5    int sum = 0;
6    for (int i = 0; i < x.length; i++) {
7      if (x[i] % 2 == 0) {
8        sum += x[i];
9      }
10   }
11   return sum;
12 }
```

a) Draw the control flow graph. Count the basic blocks and branches.

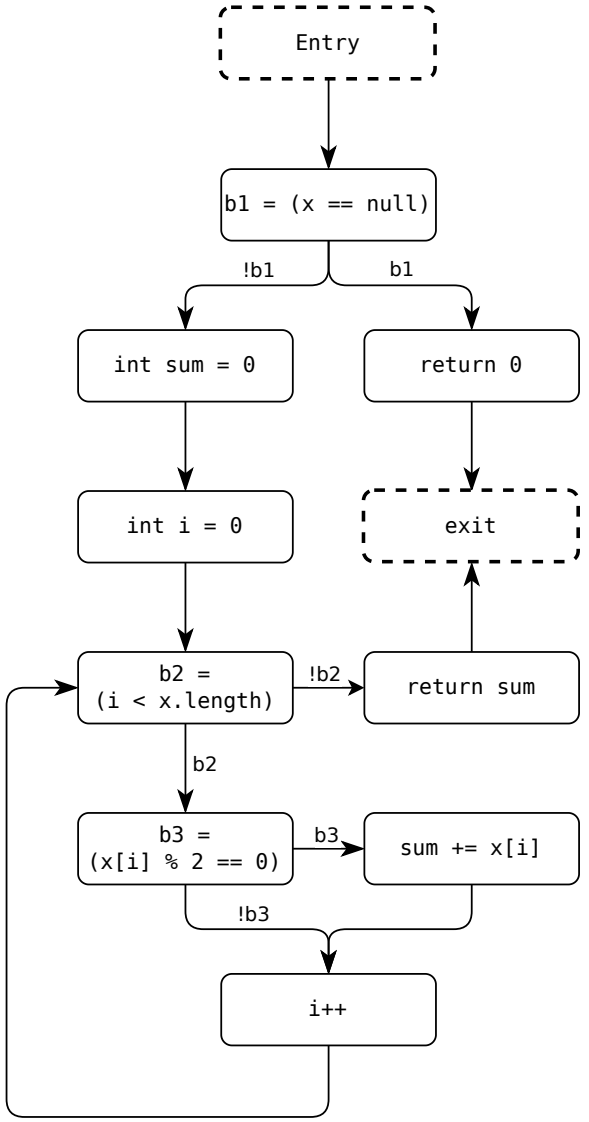
```
Basic blocks: 10
Branches: 6
```

b) Define test cases that achieve 100% basic-block coverage, but not 100% branch coverage.

```
x = null; x = {2}
```

c) Define test cases that achieve 100% branch coverage.

```
x = null; x = {2, 3}
```



Exercise 5: Condition + Branch Coverage

```
1 public String triangle(int a, int b, int c) {
2   if (a+b<c || a+c<b || b+c<a) {
3     return "invalid";
4   }
5   if (a*a+b*b==c*c || a*a+c*c==b*b || b*b+c*c==a*a) {
6     return "right␣angled";
7   }
8   return "other";
9 }
```

a) Count the number of condition values + branches.

```
Condition values: 12
Branches: 4
```

b) How much branch coverage does the test (a=1, b=1, c=1) reach?

```
Branch coverage: 2/4 = 50%
```

c) How much C+B coverage does the test (a=1, b=1, c=1) reach?

$$\text{C+B coverage: } \frac{2+6}{4+12} = 8/16 = 50\%$$

d) Construct test cases that reach 100% C+B coverage.

```
Coverage is 100% when each individual condition evaluates to
true and false at least once and each corresponding branch
statement also evaluates to true and false at least once

(a=1, b=1, c=1);  (a=1, b=2, c=3)
(a=1, b=-1, c=1); (a=0, b=1, c=-1)
```

Exercise 6: MC/DC

```
1 public int compute(int a, int b) {
2   if ((a * b == 20 || a + b == 12) && a < 10) {
3     return a;
4   } else {
5     return b;
6   }
7 }
```

a) Construct test cases that reach 100% MC/DC. List for each test case which conditions are true and which are false.

b) List the independence pair for each condition.

```
A = a < 10    B = a * b == 20    C = a + b == 12

Condition: (A && (B || C))

 # | A | B | C | Res
---+---+---+---+-----
 1 | F | F | F |  F
 2 | F | F | T |  F
 3 | F | T | F |  F
 4 | F | T | T |  F
 5 | T | F | F |  F
 6 | T | F | T |  T
 7 | T | T | F |  T
 8 | T | T | T |  T

IndependencePairs = {{2,6}, {3,7}, {4,8}} ∪ {{5,7}} ∪ {{5,6}}
Tests = {3, 5, 6, 7}

3: (a=20, b=1)
5: (a=1, b=1)
6: (a=3, b=9)
7: (a=5, b=4)
```

Exercise 7: DU-Pairs Coverage

```
1  public int range(int a, int b, int c) {
2    int max = a;
3    int min = a;
4    if (a < b) {
5      max = b;
6    } else {
7      min = b;
8    }
9    if (max < c) {
10     max = c;
11   }
12   if (c < min) {
13     min = c;
14   }
15   return max - min;
16 }
```

a) List all DU pairs for variables `max` and `min`.

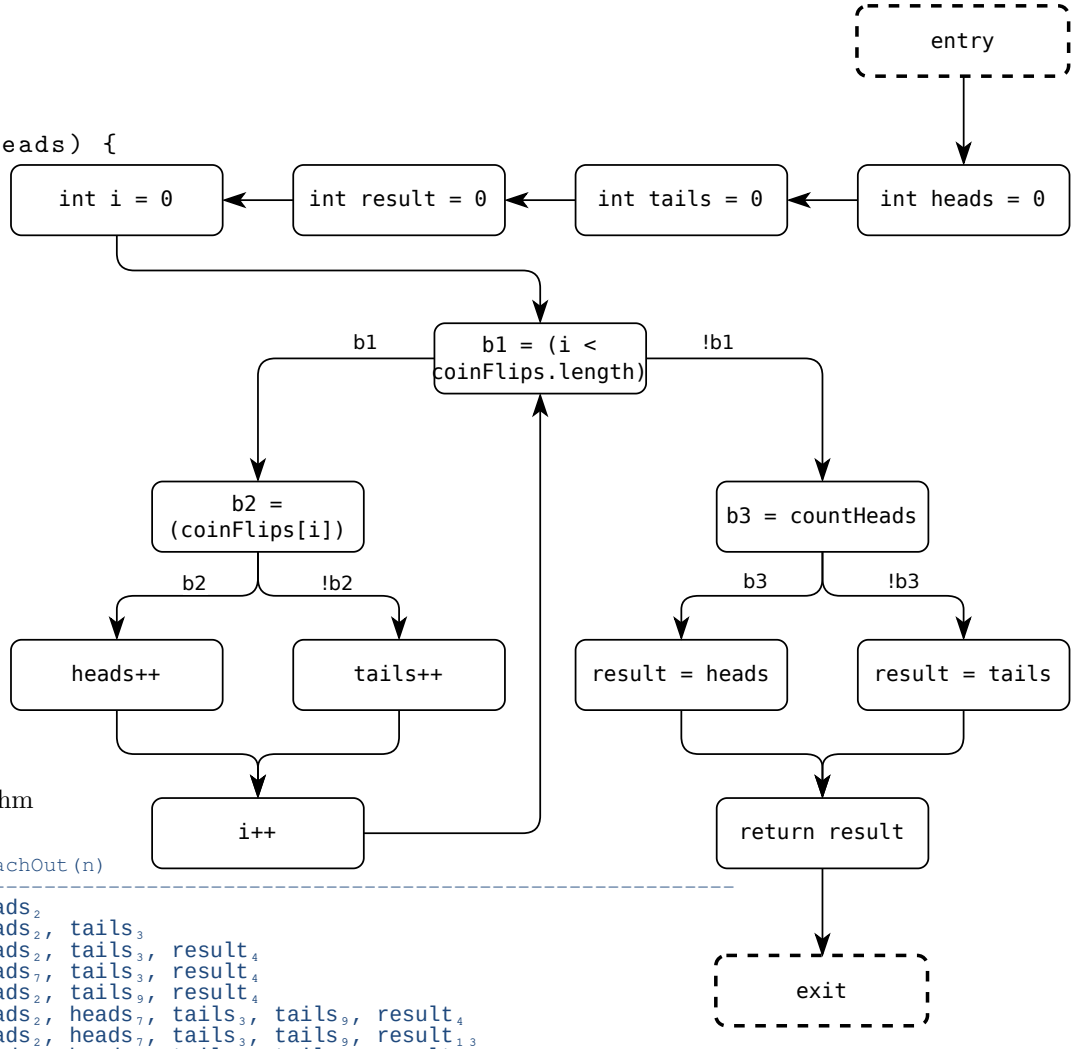
```
DUpairs_max = {{2,9}, {2,15}, {5,9}, {5,15}, {10,15}}
DUpairs_min = {{3,12}, {3,15}, {7,12}, {7,15}, {13,15}}
```

b) Construct test cases that reach 100% DU-pairs coverage. For each test, list all DU pairs it covers.

```
{{2,9}, {2,15}, {7,12}, {7,15}}: (a=0, b=0, c=0)
{{5,9}, {5,15}, {3,12}, {3,15}}: (a=0, b=1, c=0)
{{10,15}}: (a=0, b=0, c=1)
{{13,15}}: (a=0, b=0, c=-1)
```

Exercise 8: Measuring DU-Pairs Coverage

```
1  public void countFlips(boolean[] coinFlips, boolean countHeads) {
2    int heads = 0;
3    int tails = 0;
4    int result = 0;
5    for (boolean isHeads: coinFlips) {
6      if (isHeads) {
7        heads = heads + 1
8      } else {
9        tails = tails + 1;
10     }
11   }
12   if (countHeads) {
13     result = heads;
14   } else {
15     result = tails;
16   }
17   return result;
18 }
```



a) Draw the control flow graph for function `countFlips` and apply the algorithm for computing reaching definitions for variables `heads`, `tails` and `result`.

n	Reach(n)	ReachOut(n)
2	-	$heads_2$
3	$heads_2$	$heads_2$, $tails_3$
4	$heads_2$, $tails_3$	$heads_2$, $tails_3$, $result_4$
7	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$	$heads_7$, $tails_3$, $result_4$
9	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$	$heads_2$, $tails_9$, $result_4$
12	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$
13	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_{13}$
15	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_{15}$
17	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$, $result_{13}$, $result_{15}$	$heads_2$, $heads_7$, $tails_3$, $tails_9$, $result_4$, $result_{13}$, $result_{15}$

b) List the DU pairs for variables `heads`, `tails` and `result`.

DUpairs(heads) = {(2,7), (7,7), (2,13), (7,13)}
DUpairs(tails) = {(3,9), (9,9), (3,15), (9,15)}
DUpairs(result) = {(4,17), (13,17), (15,17)}

c) 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 (`coinFlips=[true, true]`, `countHeads = false`)? You may assume the maps start freshly initialized.

```
defCover = {"heads": 7, "tails": 3, "result": 15}
useCover = {"heads": {"2": {"7": 1}, "7": {"7": 1}}, "tails": {"3": {"15": 1}}, "result": {"15": {"17": 1}}}
```

Exercise 10: Test Doubles

Classify the following objects into one of the five kinds of test doubles.

a) An external API server that returns pre-defined responses and verifies that specific requests were made during testing.

`Mock`

b) A database connection wrapper that records every query made to a particular table.

`Spie`

c) A database connection that returns pre-defined data for specific queries.

`Stub`

d) A logger that does not perform any logging and is only used to fulfill a method requirement.

`Dummy`

e) A file system that emulates the behavior of a real file system without actually writing to disk.

`Fake`

f) An HTTP server that returns pre-defined responses to specific requests.

`Stub`

g) An email service that captures and stores outgoing emails and triggers pre-defined incoming email events.

`Spie`

h) A database connection that ignores all operations and is not used during testing.

`Dummy`

i) A logger that records information about logged messages during testing and checks for the existence of certain string patterns.

`Spie`

j) A data prediction unit that, in contrast to its production implementation, uses a simplified algorithm to decrease the runtime of the tests.

`Fake`