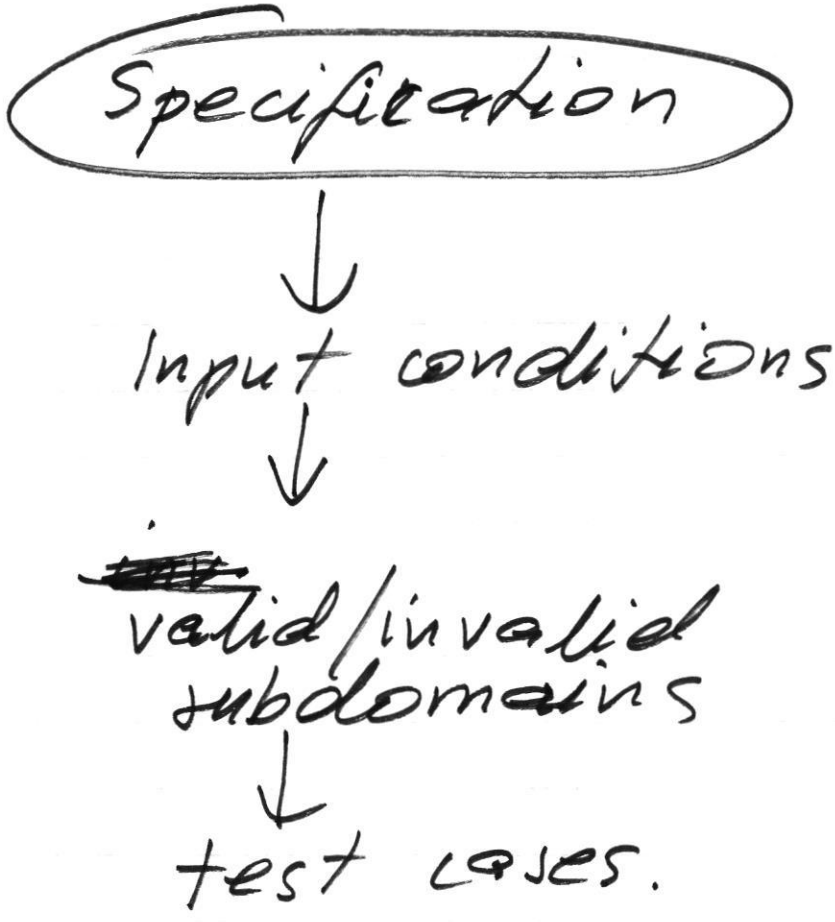
Homework #1 is
posted

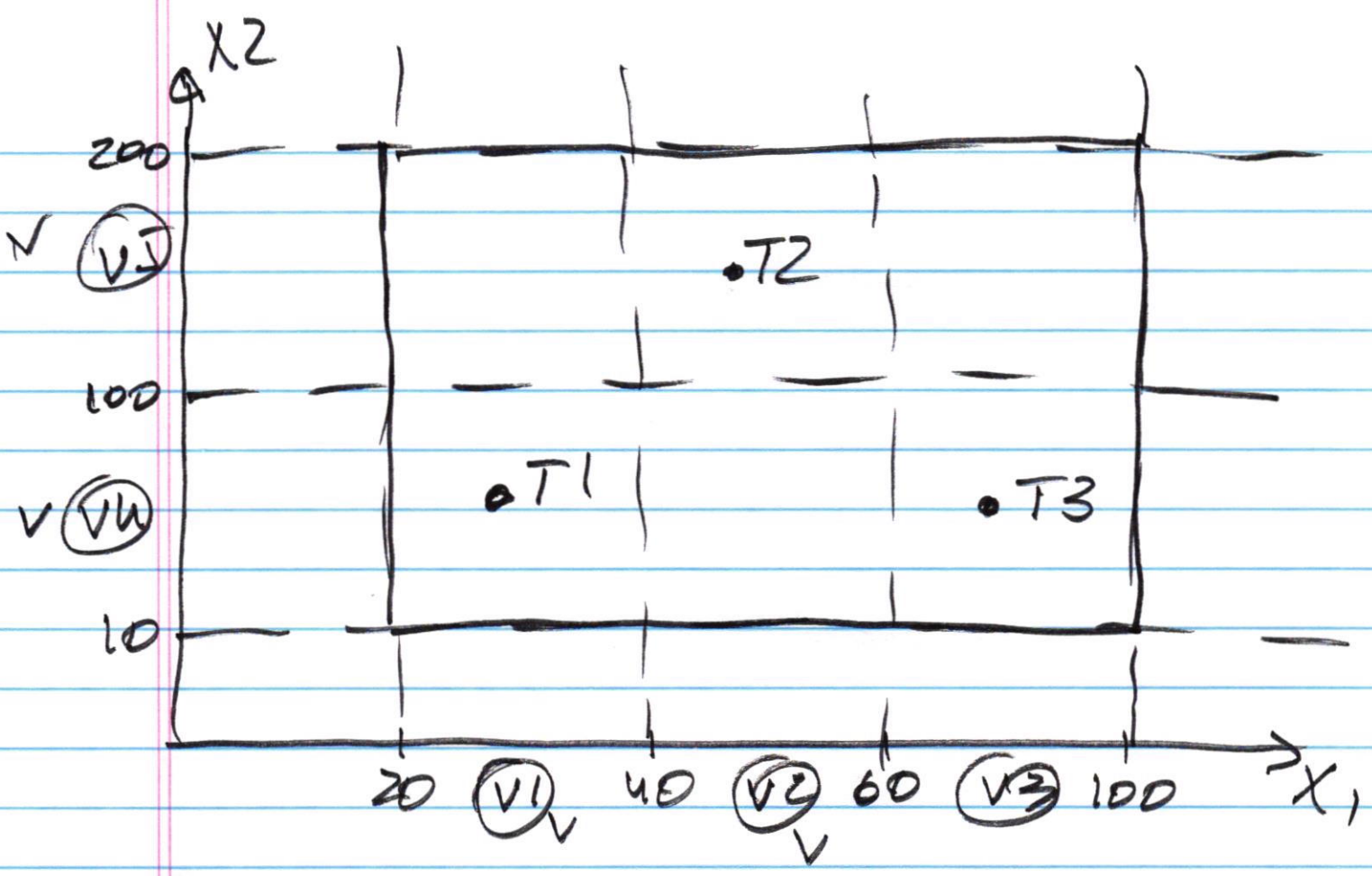
# Specification-based testing.



# Equivalence Partitioning testing.

1. Normal Eq. testing.
   select tests from valid subdomains.
   (a) Weak normal Eq. testing
   1-dim subdomains
   (b) strong normal Eq. testing.
   n-dim subdomains
   (n>1)

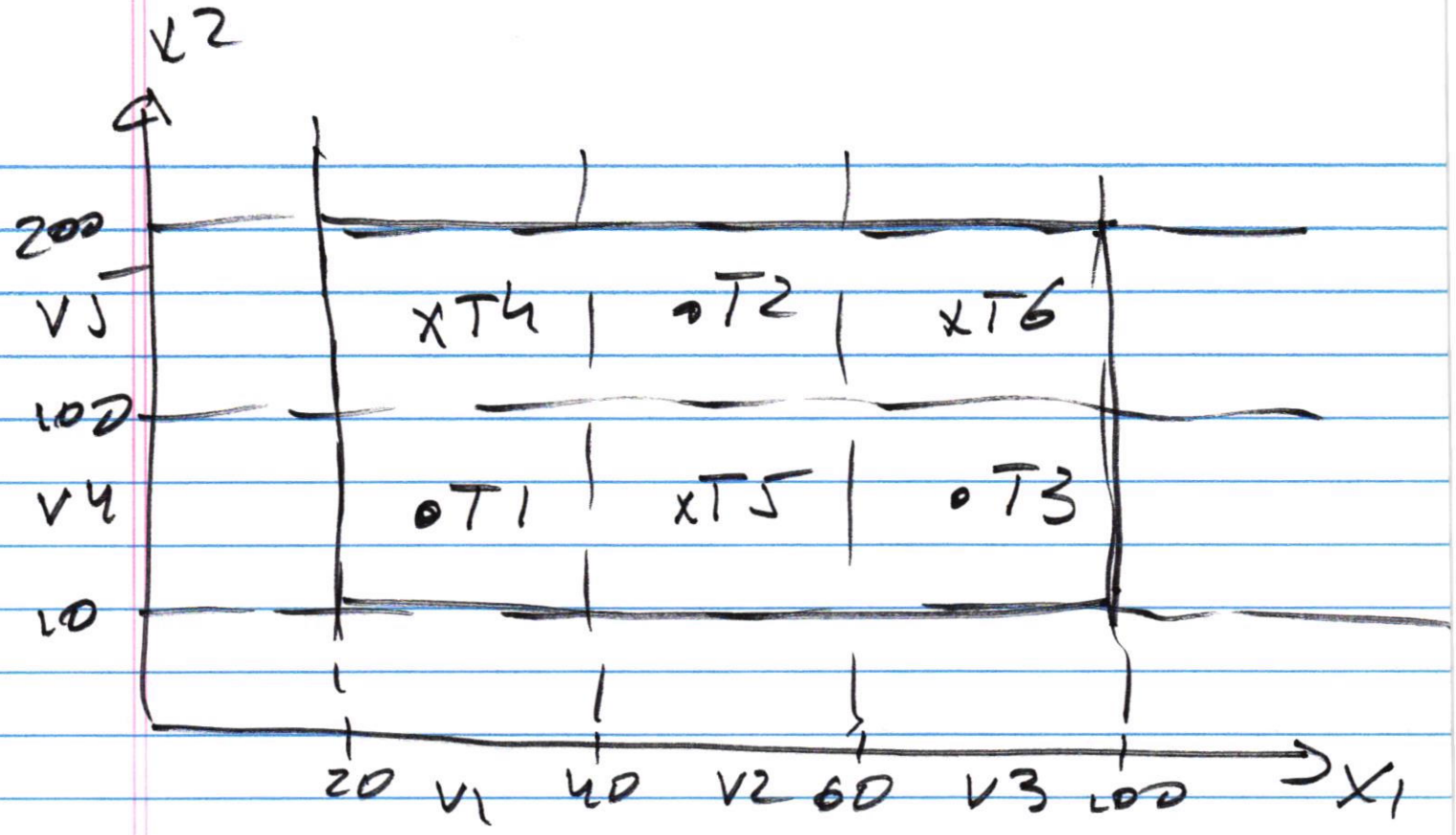
2. Robust Eq. testing.
   select tests from invalid subdomains
   (a) weak robust Eq. testing.
   1-dim invalid subdomains
   (b) strong robust Eq. testing.
   n-dim invalid subdomains




X1: V1, V2, V3
X2: V4, V5

## Weak Normal Eq. tests

T1: X1 = 25, X2 = 70 (V1, V4)
T2: X1 = 51, X2 = 120 (V2, V5)
T3: X1 = 75, X2 = 45 (V3)




Strong Normal Eq. tests

2-elim valid subdomains

(V1,V4), (V1,V5), (V2,V4), (V2,V5), (V3,V4), (V3,V5)

T1 : (V1, V4)
T2 : (V2, V5)
T3 : (V3, V4)
T4 : x1=25, x2=125 (V1, V5)
T5 : x1=50, x2=20 (V2, V4)
T6 : x1=25, x2=120 (V3, V5)

# Robust Eq. testing.

(a) weak robust eq. testing.

select tests from 1-dim invalid subdomains.

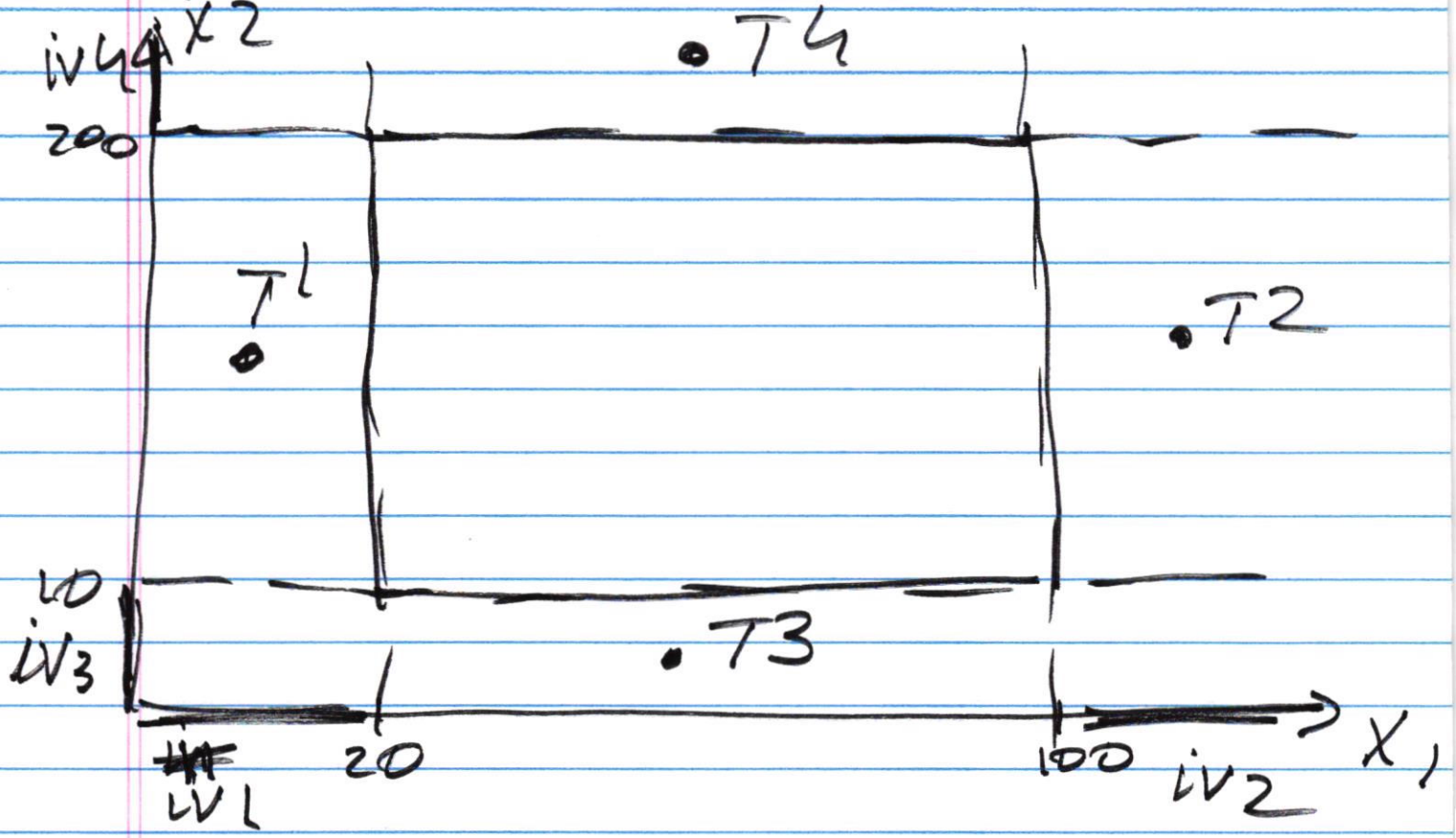
a test case should have one invalid value and all the remaining values should be valid

$$x_1, x_2, x_3, \ldots, x_n$$

(1) only $x_i$ is invalid

(2) all remaining $x_k$ $(k \neq i)$ should have valid values

# Weak Robust Eq. testing.




X1: 2 invalid subdomains: iV1, iV2

X2: —— " " —— : iV3, iV4

X1

T1: X1 = 10, X2 = 120 (iV1)

T2: X1 = 145, X2 = 125 (iV2)

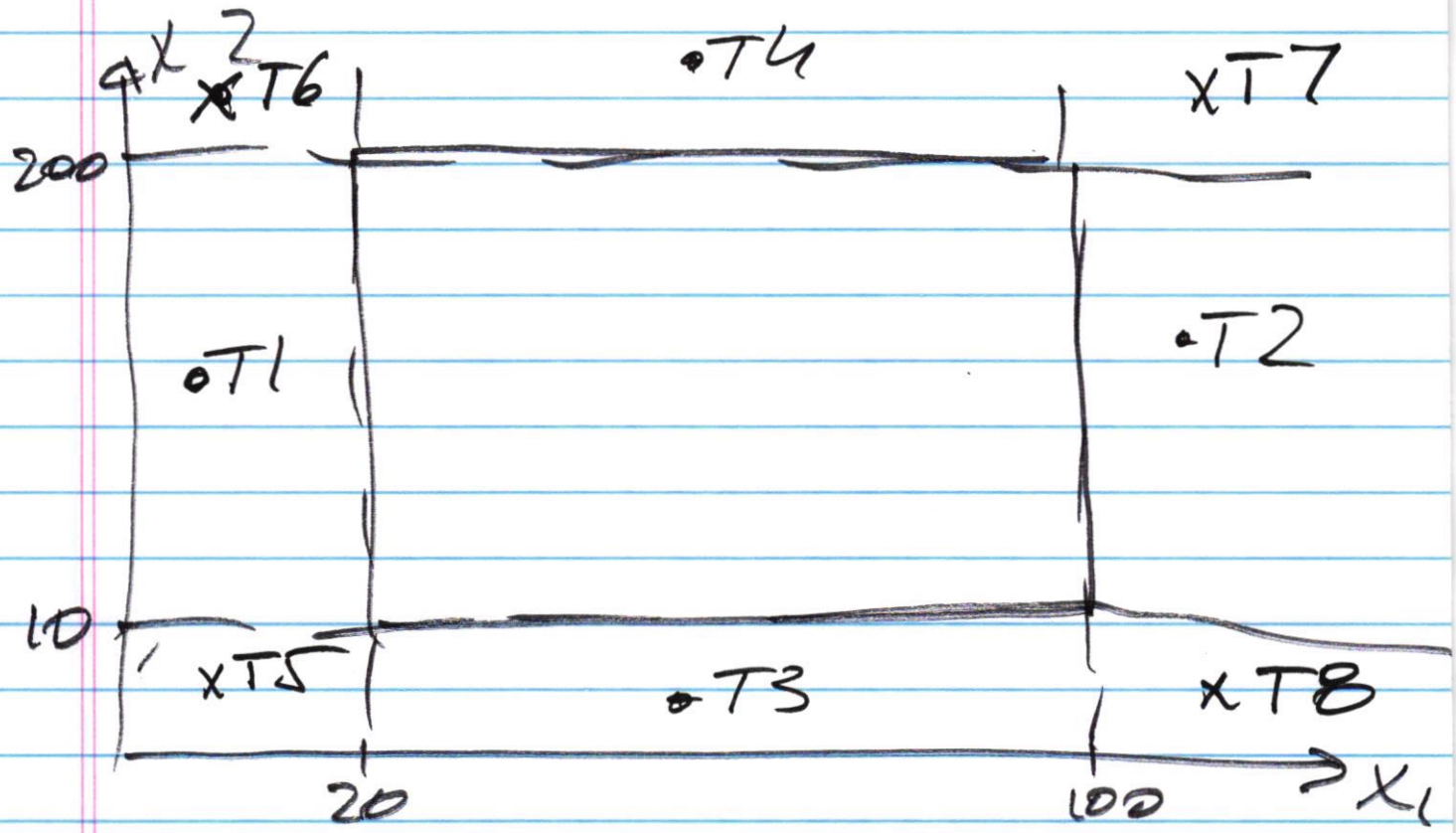
X2:

T3: X1 = 75, X2 = 5 (iV3)

T4: X1 = 70, X2 = 210 (iV4)

# Strong robust Eq. testing.

testing multi-dim invalid subdomains.




Week: T1, T2, T3, T4

strong: T1, T2, T3, T4, T5, T6, T7, T8

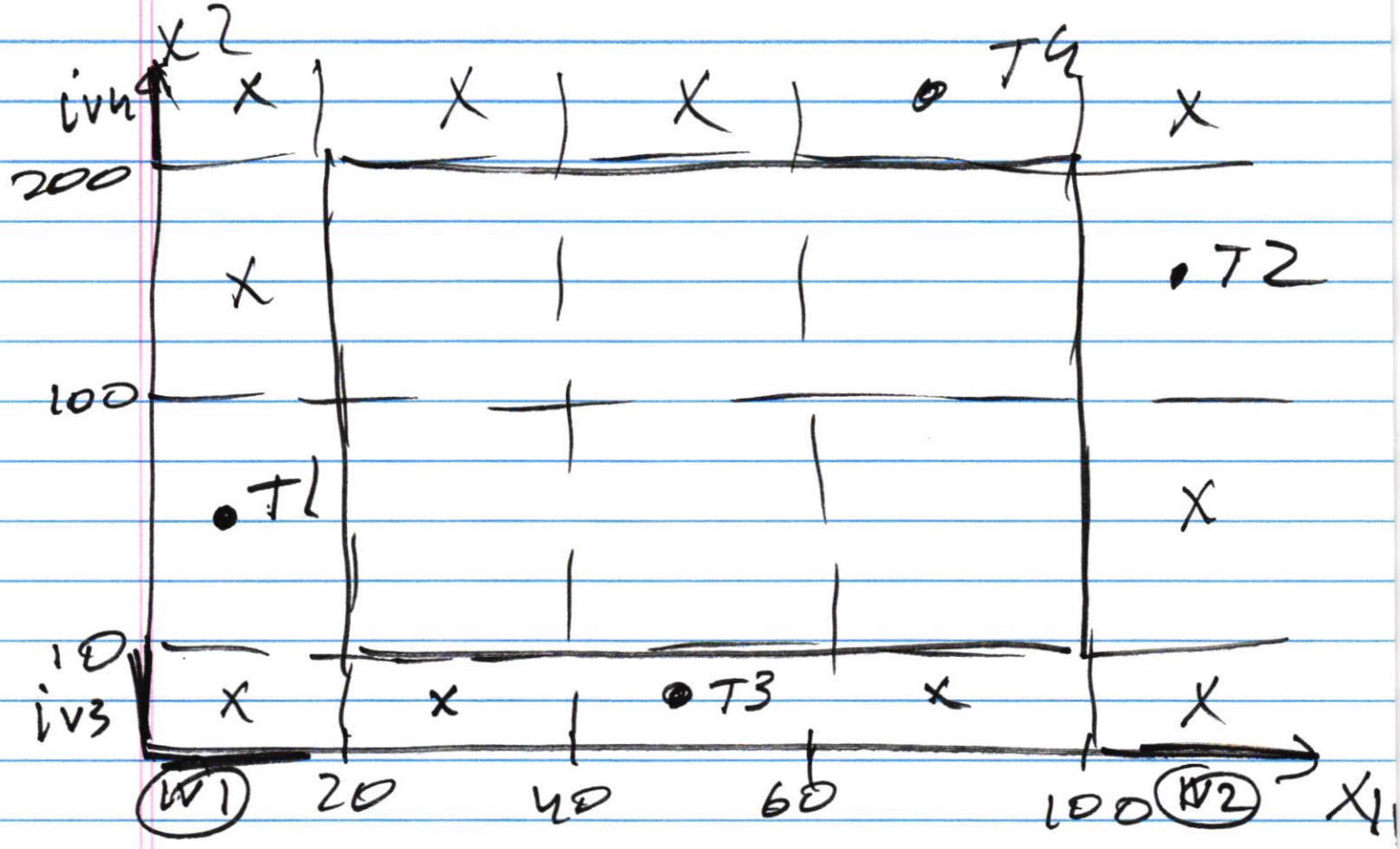
T5: $x_1 = 16$, $x_2 = 5$

T6: $x_1 = 10$, $x_2 = 205$

T7: $x_1 = 107$, $x_2 = 210$

T8: $x_1 = 110$, $x_2 = 6$

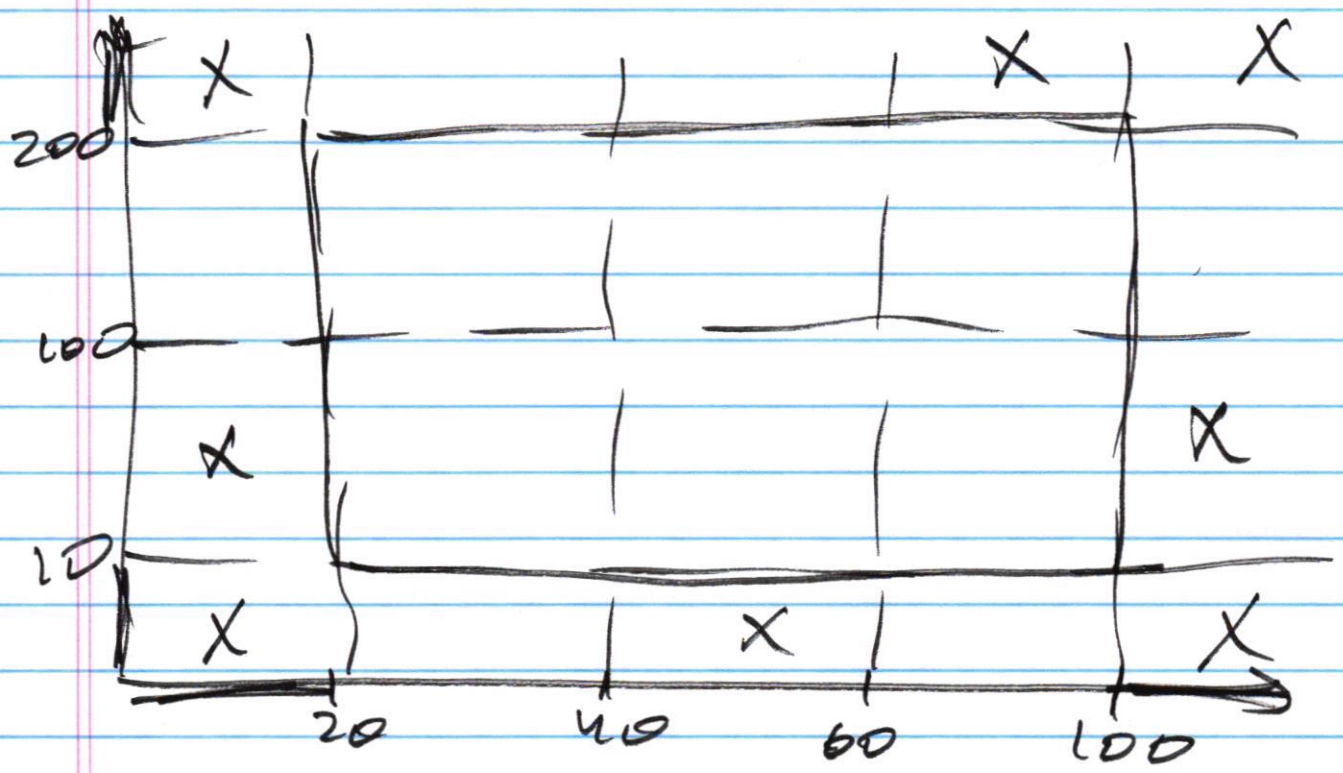
# Strong robust Eq. testing.




• weak: T1, T2, T3 Th

x strong: 1h tests.

# another version




strong: 8 tests.

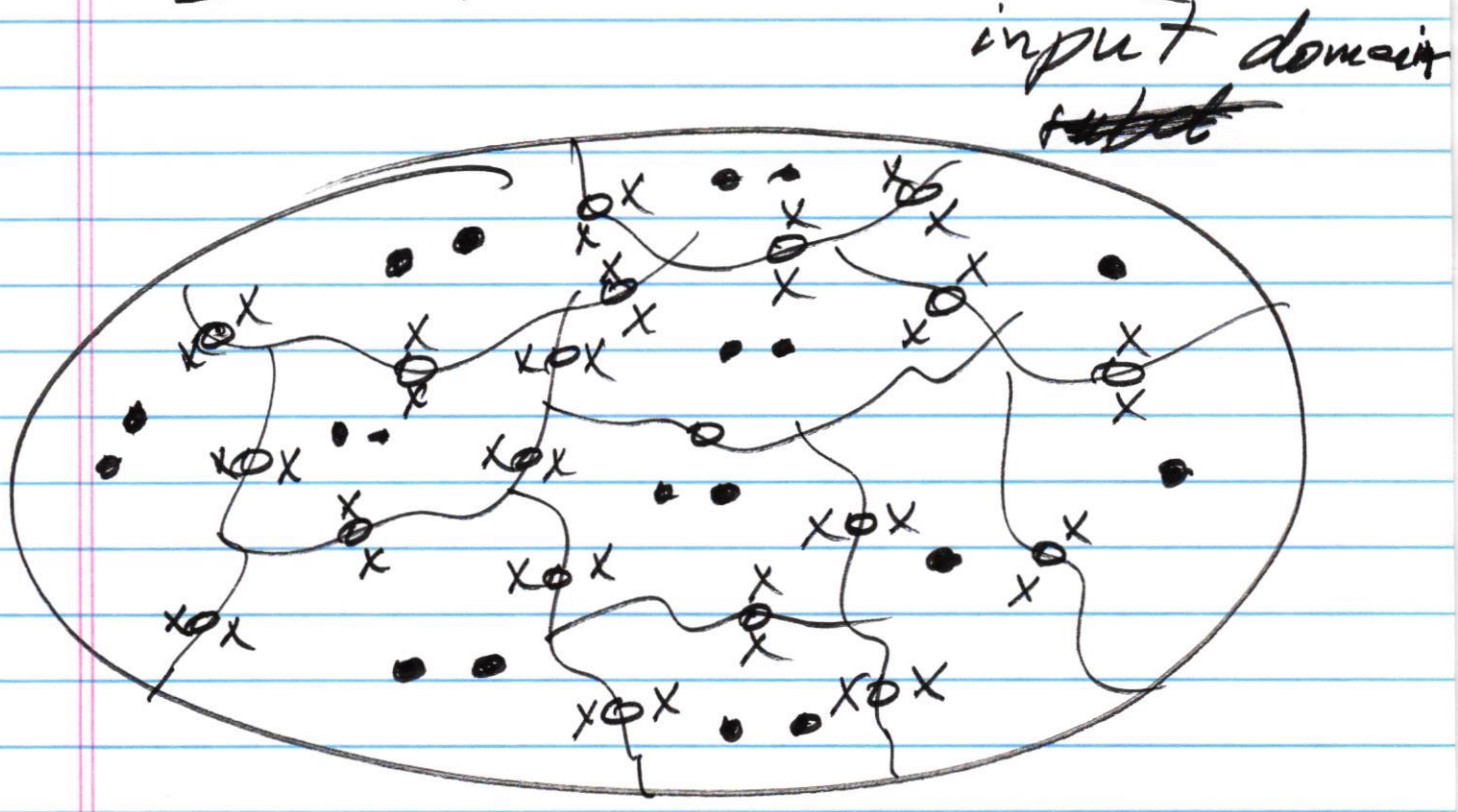
# Testing boundaries

it complements normal equivalence testing.

## Idea

1. select tests on boundaries between subdomains
2. select tests "around" boundaries between subdomains.

# Testing boundaries




o tests on boundaries

x tests around boundaries

# Testing boundaries

1. Normal Boundary Value Analysis Testing (BVA)

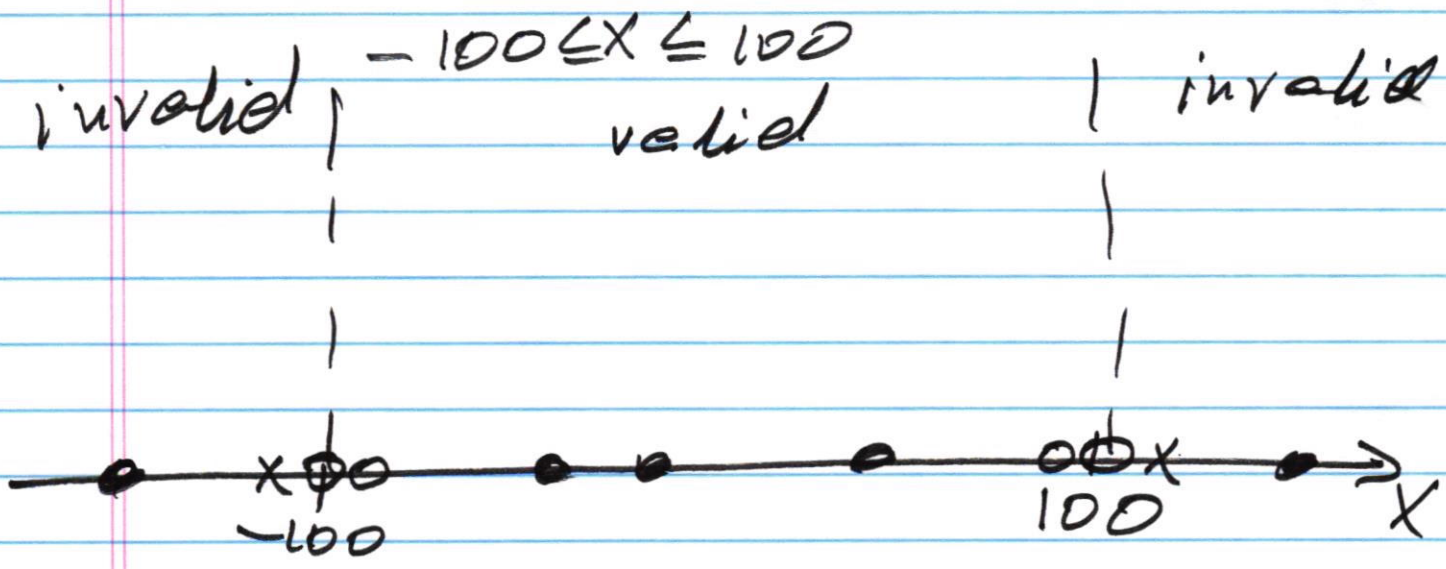
select valid boundary values

2. Robust boundary value testing.

select invalid boundary values

# input condition

x : integer




o **Normal BVA tests**

**on boundaries**

T1 : x = 100

T2 : x = -100

**around boundaries**

T3 : x = 99

T4 : x = -99

x **Robust boundary tests**

T5 : x = 101

T6 : x = -101

x: float

Δ selected by tester/developer

Δ = 0.01

## Normal BVA tests

### on boundaries

T1: x = 100
T2: x = -100

### around boundaries

T3: x = 99.99
T4: x = -99.99

## Robust boundary tests

T5: x = 100.01
T6: x = -100.01

Input file contains
N records

$1 \leq N \leq 255$

boundaries:

$N = 1, \quad N = 255$

## Normal boundary tests

### on boundaries

+1: 1 record
+2: 255 records

### around boundaries

+3: 2 records
+4: 254 " "

## Robust boundary tests

+5: N = 0 empty file
+6: N = 256 records.

# A set of values

vehicle type: bus, truck,
taxi cab, passenger

boundaries: ?

no boundaries... !!!

---

"must be" condition

"1st character of a variable must be a letter"

no boundaries, !!!

# input condition

x: float

$-100 \leq x \leq 100$


- Normal boundary tests

  on boundaries $\Delta = 0.01$

  T1: $x = -100$

  T2: $x = 99.99$

  around boundaries

  T3: $x = -99.99$

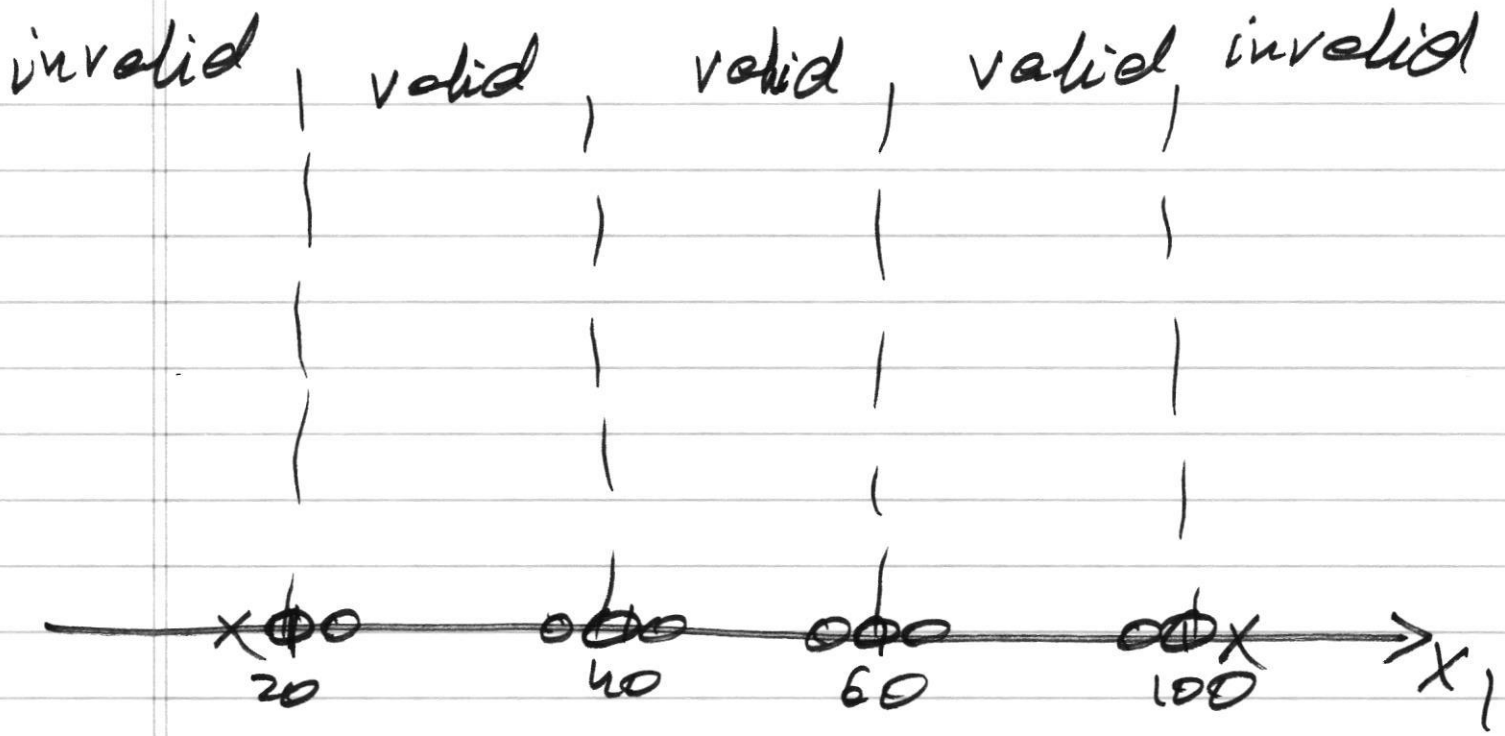
  T4: $x = +99.98$ (?)

- Robust boundary tests

  T5: $x = 100$

  T6: $x = 100.01$

  T7: $x = -100.01$




3 valid subdomains

2 invalid — " —

o Normal boundary tests 10 tests

x Robust boundary tests 2 tests

# Integer array dimension declaration

int n(d[d]...)

n is the symbolic name of the array

d is a dimension declaration

symbolic name can have 1-6 letters or digits, where the first character must be a letter

The minimum # of dimensions = 1

The maximum # of dimensions = 7

The format of a dimension declaration [lb:]ub

lb: lower bound; ub upper bound

The bounds may be in the range -65,534 to 65,535

if lb is not specified, lb=1 ub $\geq$ lb

| input conditions | valid sub-domain(s) | invalid sub-domain(s) |
|---|---|---|
| size of array name ✓ | 1 - 6 (1) | 0; (2) > 6 (3) |
| array name has letters/digits ✗ no boundaries | has letters (4)<br>has digit (5) | something (6)<br>else |
| array name starts with a letter no boundaries ✗ | yes (7)<br>. | no (8)<br>. |
| # of dimensions ✓ | 1 - 7 (9) | 0; (10) > 7 (11) |
| lower bound ✓ | -65,534 - +65,535 (12) | <-65,534 - (13) >+65,535 (14) |
| upper bound ✓ | -65,534 - +65,535 (15) | <-65,534 - (16) >+65,535 (17) |
| lower bound specified ✗ no boundaries | yes (18) no (19) | |
| upper bound to lower bound ✓ | ub > lb (20)<br>ub = lb (21) | ub < lb (22) |

# Homework #1

# HOMEWORK ASSIGNMENT #1

CS589; Fall 2021
Due Date: **September 22, 2021**
Late homework 50% off
After **September 26,** the homework assignment will not be accepted.

This is an **individual** assignment. **Identical or similar** solutions will be penalized.

**Submission:** All homework assignments must be submitted on the Blackboard. The hardcopy submissions will not be accepted.

## SPECIFICATION-BASED TESTING

Suppose a software component (called a Car Insurance System) has been implemented to handle processing the annual renewal of a hypothetical auto insurance policy. The following are requirements for the component:

> If the insured made one claim in the last year and is age 25 or older, the increase is $60 and no letter is sent. If the insured had no claims in the last year and is age 24 or younger, the increase is $60 and no letter is sent. If the insured had no claims in the last year and is age 25 or older, the increase is $35 and no letter is sent. If the insured made two, three, or four claims in the last year and is age 24 or younger, the increase is $310 and a warning letter is sent. If the insured made one claim in the last year and is age 24 or younger, the increase is $110 and a warning letter is sent. If the insured made two, three, or four claims in the last year and is age 25 or older, the increase is $210 and a warning letter is sent. If the insured made five or more claims in the last year, the policy is canceled.

The component accepts the input in the following format (6 input variables):

last name
first name
Age
car type
VIN
# of claims

The maximum size of the "*first name*" is 10 characters and "*last name*" is 15 characters. The *car type* can be: *sedan, mini-van*, *truck, SUV, Taxi*

Assumptions:

- *last name* and *first name* contain only letter characters.
- *age* is an integer. The minimum *age* is 18 and the maximum *age* is 110.
- The maximum *# of claims* is 10.
- VIN (a vehicle identification number) contains 17 characters (letters and digits) and has the following format: DLLLLDDLLLLDDDDDD, where D is a digit and L is a letter.

A sample component test:
Test #1:

| | |
|---|---|
| last name | Smith |
| first name | John |
| Age | 27 |
| car type | Truck |
| VIN | 1HGBH41JXMN109186 |
| # of claims | 3 |

**PROBLEM #1** (35 points): Equivalence partition testing
For the Car Insurance System identify input conditions related to:

1. last name
2. first name
3. Age
4. Car type
5. VIN
6. # of claims

From the identified input conditions list valid and invalid sub-domains (equivalence classes). Based on identified sub-domains design test cases using:

a. Strong normal equivalence testing.
b. Weak robust equivalence testing.

Hint: Before designing test cases, identify related/unrelated input conditions.

**PROBLEM #2** (30 points): Boundary-value testing
Based on identified sub-domains in Problem #1 design:

1. Normal Boundary-Value Analysis test cases.
2. Robust Boundary test cases.

**PROBLEM #3** (35 points): Decision-Table based testing

Suppose a software component (called a Grader component) has been implemented to automatically compute a grade in a course. A course taught at a university has two components: (1) two exams and (2) a project. To pass the course with grade C a student must score at least 50 points in Exam-1, 60 points in Exam-2, and 50 points in the Project. Students pass the course with grade B if they score at least 60 points in Exam-1, 65 points in Exam-2, and 60 points in the Project. If, in addition to this, the average of the exams and the Project is at least 75 points then students are awarded a grade A. Final grades for the course are: A, B, C, and E. The Grader component accepts four inputs:

Student #
Exam-1
Exam-2
Project

**Assumptions:**

- Assume *Exam-1, Exam-2,* and *Project* are integers and represent the scores of the exams and the project.
- The ranges for the exam scores and the project score are:
  - $0 \leq Exam\text{-}1 \leq 100$
  - $0 \leq Exam\text{-}2 \leq 100$
  - $0 \leq Project \leq 100$
- *Student #* is a number represented as a 9-character string in the following format: AXXXXXXXX, where X is a digit.

**Sample test cases for the Grader component:**

Test #1: *Student #*=A11112222, *Exam-1*=57, *Exam-2* = 64, *Project*=55
Test #2: *Student #*=A42312242, *Exam-1*=75, *Exam-2* = 24, *Project*=85

Use **decision-table** based testing to design test cases to test the GRADER program. Provide a decision table and test cases derived from the decision table.

**Note:** In your solution conditions cannot be complex logical expressions. For example:

$$(Exam\text{-}1 < 50) \text{ and } (Exam\text{-}2 \geq 60)$$

is **not acceptable** as a condition in the decision table. However, "$Exam\text{-}1 < 50$" is a condition; "$Exam\text{-}2 \geq 60$" is a condition.