Homework #1 is posted

cause-effect graphing.

related to decision-table testing.

specification

↓

input conditions

+

actions

⇓

cause-effect graph

⇓

decision table

⇓

test cases

cause-effect graph

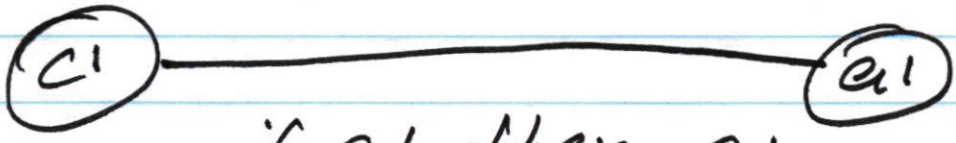
identify logical relationships between input conditions (causes)

and

actions (effects)

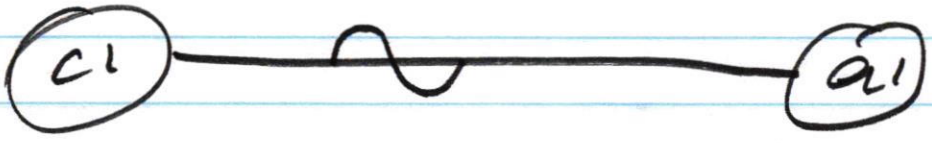
cause-effect graph

1. identity relationship



if c1 then a1

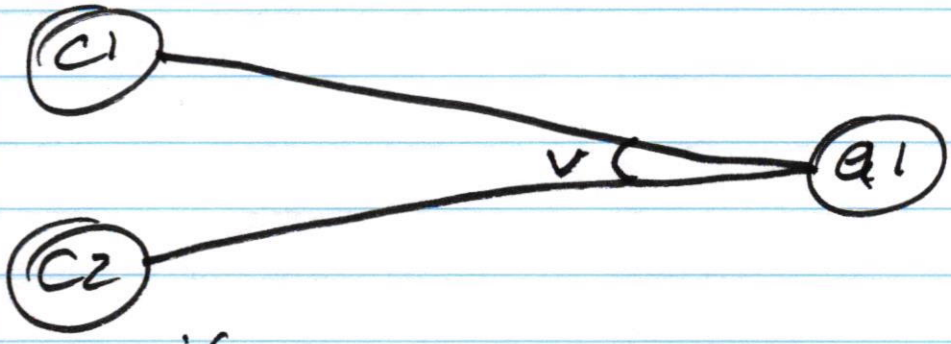
2. "not" relationship



if not c1 then a1

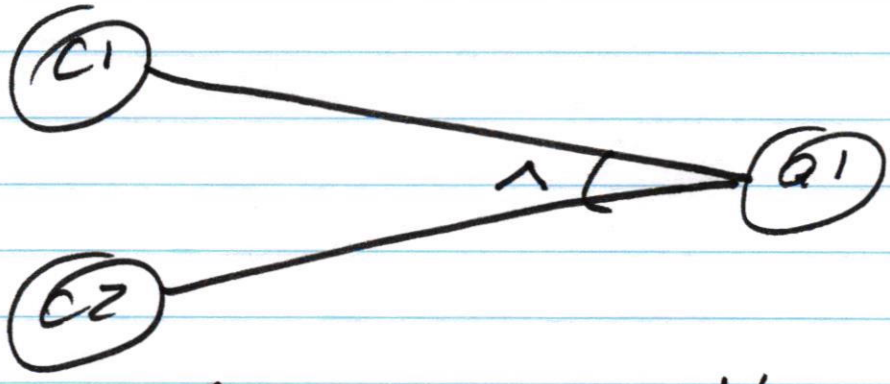
3. no relationship

4. "or" relationship



if C1 or C2 then a1

5. "and" relationship



if C1 and C2 then a1

DISPLAY p n

This command displays "n" elements from a list starting from position p

List: 1, 8, 4, 5, 2, 1

DISPLAY 4 2

5, 2 are displayed.

---

actions

1). a subhit is displayed

2) syntax error msg.

3) incorrect values of parameters msg.

input conditions

c1: the first 7 characters are "DISPLAY"

c2: p is an integer

c3: $1 \leq p \leq$ size of the list

c4: n is integer

c5: $n \geq 1$

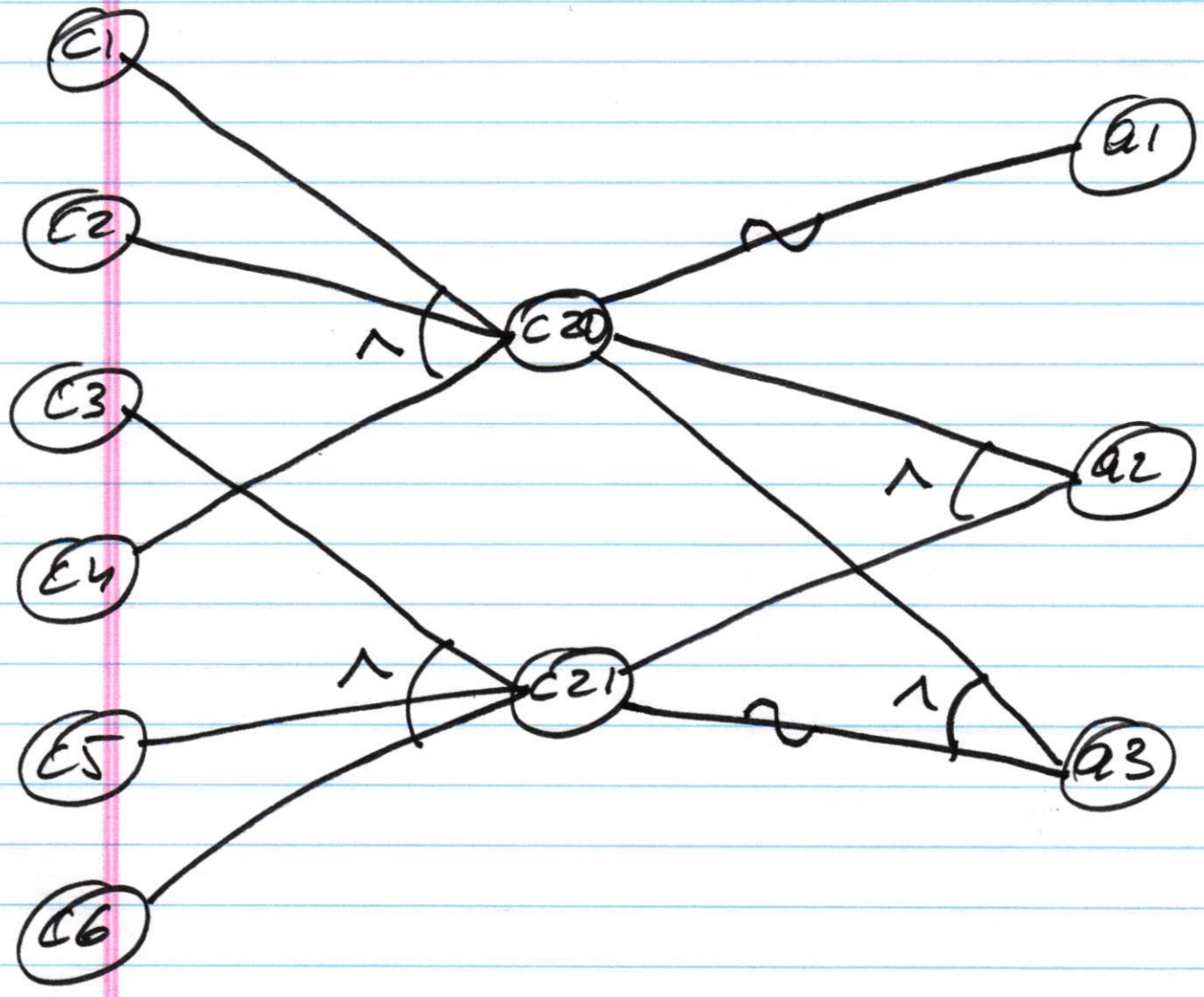
c6: $p + n \leq$ size of the list + 1

$6 + 1 \leq 6 + 1$

---

actions:

a1: invalid syntax error msg.
a2: sublist is displayed
a3: incorrect values of parameters.



C20 syntax is correct

C21 correct values of parameters.

create decision table

identify rules

for each action identify all combinations of conditions that "influence" the action.

	R1	R2	R3	R4	R5	R6	R7
C1	F	T	T	T	T	T	T
C2	T	F	T	T	T	T	T
C3	T			T	F	T	T
C4	T	T	F	T	T	T	T
C5				T	T	F	T
C6				T	T	T	F
Q1	X	X	X				
Q2				X			
Q3					X	X	X

Test cases

R1:T1: DISPLAB 5 8

R2:T2: DISPLAY A 8

R3:T3: DISPLAY 5 B

R4:T4: DISPLAY 9 10

R5:T5: DISPLAY 120 5

R6:T6: DISPLAY 40 0

R7:T7: DISPLAY 80 25

Assume the list has 100 elements.

1. Specification-based testing.

2. code-based testing.

⇓

source code is used to design test cases

Code-based testing.

- (*) statement testing.
- (*) branch —"— ✓
- * multiple-condition —"—
- * data flow —"—
- * path —"—
- A ...

statement testing.

Idea: every statement should be executed at least once during testing.

Statement:

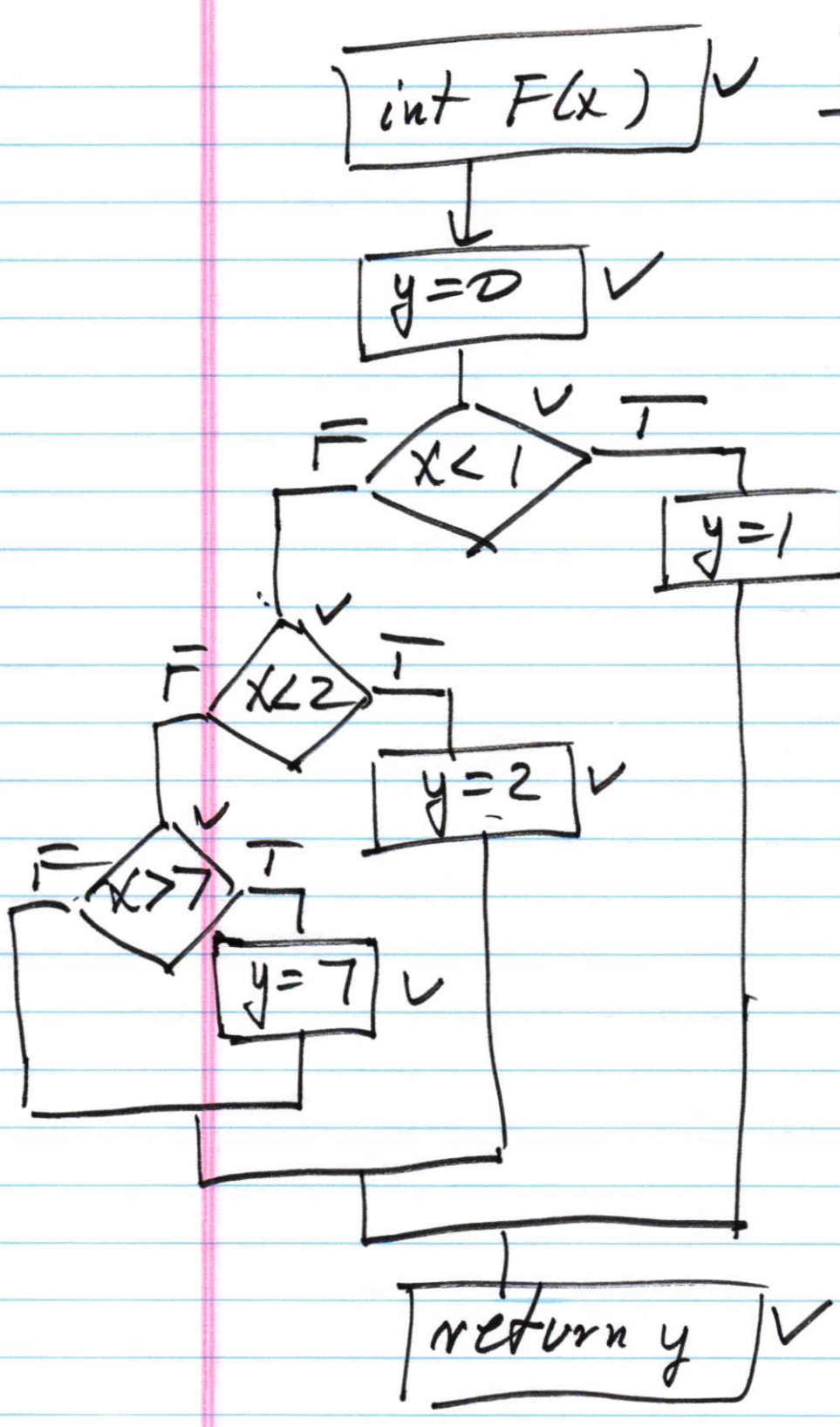
* assignment statement
* input/output — " —
* return — " —
* predicate of conditional statement
  - -if- statement
  - -while- statement
  -

variable declaration

int x ; not a statement

```
int F(int x ) { //entry statement
  int y;
  y = 0;
  if (x<1) { y = 1; }
  else {
      if (x<2) { y = 2; }
      else {
          if (x>7) y = 7;
      }
  }
  return y;
} //exit statement
```

9 statements



Tests:

T1: x=0

T2: x=1

T3: x=8

All statements are executed

Test coverage 100%

Branch testing.

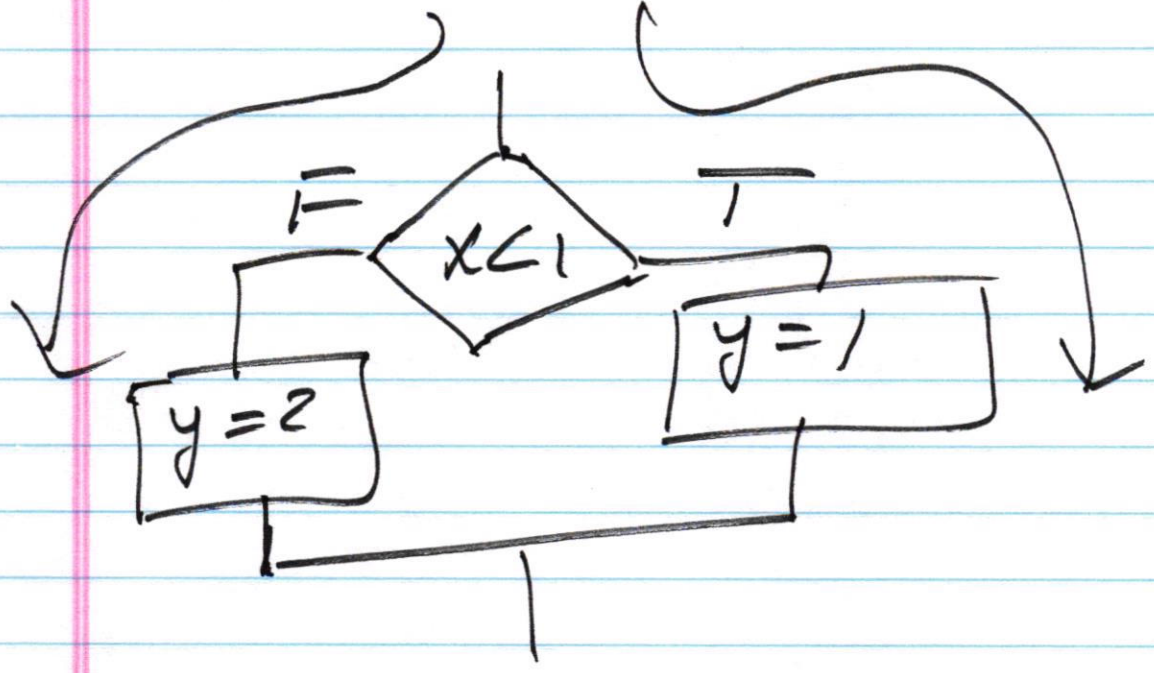
Idea: every branch in the source code should be executed at least once.

conditioned statements:

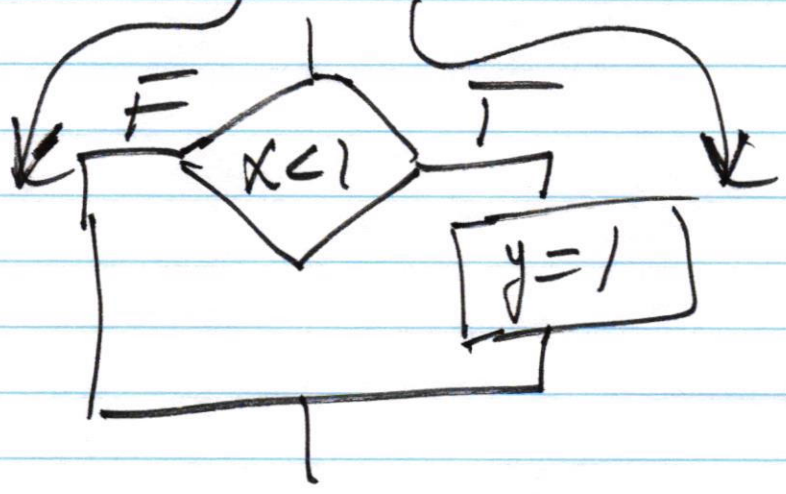
- if - statement
- while - " -
- switch - " -
-

if - statement

```
if (x<1)   y=1
else y=2;
```

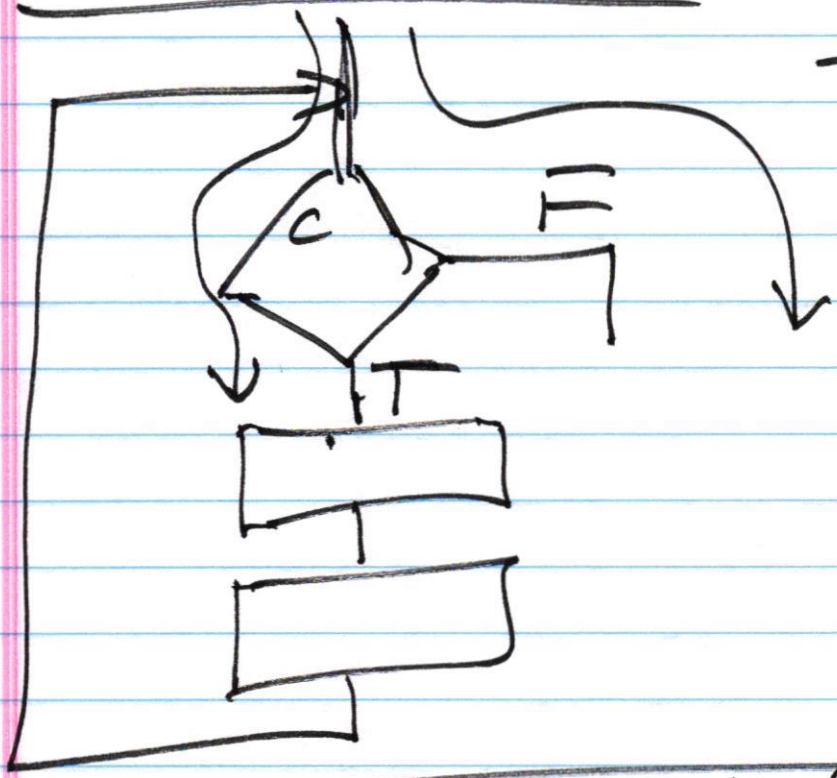



```
if (x<1) y=1;
```

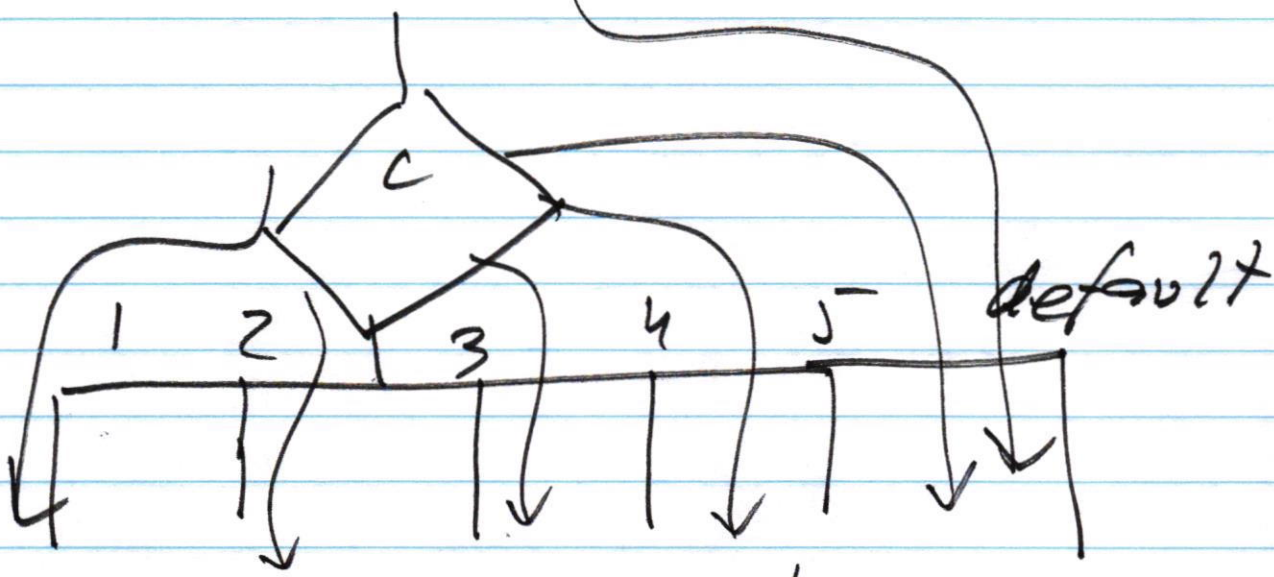


while-loop

2 branches



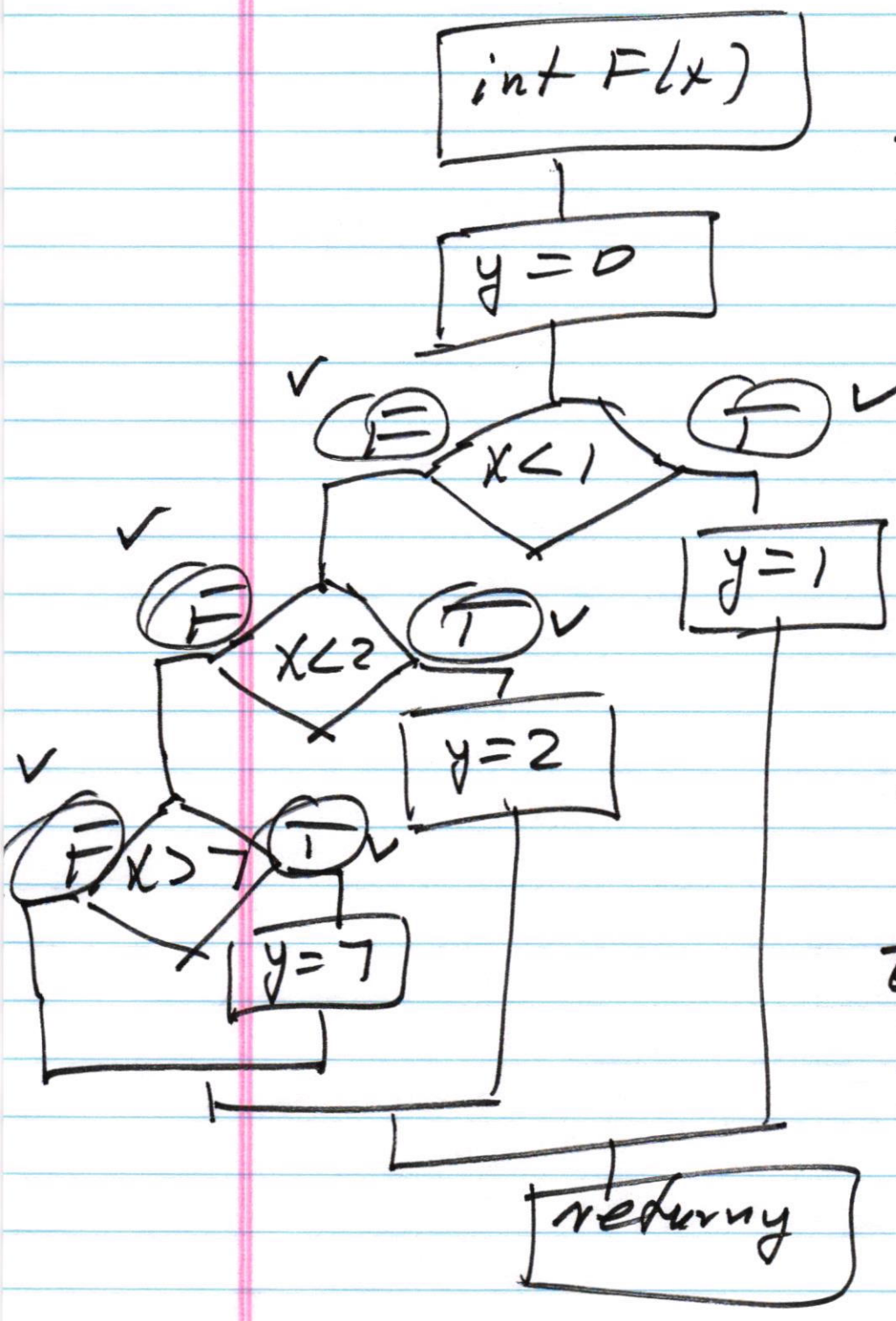
switch-statement



6 branches

branch testing.

6 branches



T1: x=0

T2: x=1

coverage:

$\frac{3}{6} \cdot 100\% = 50\%$

T3: x=8

$\frac{5}{6} \cdot 100\% = 83\%$

T4: x=5

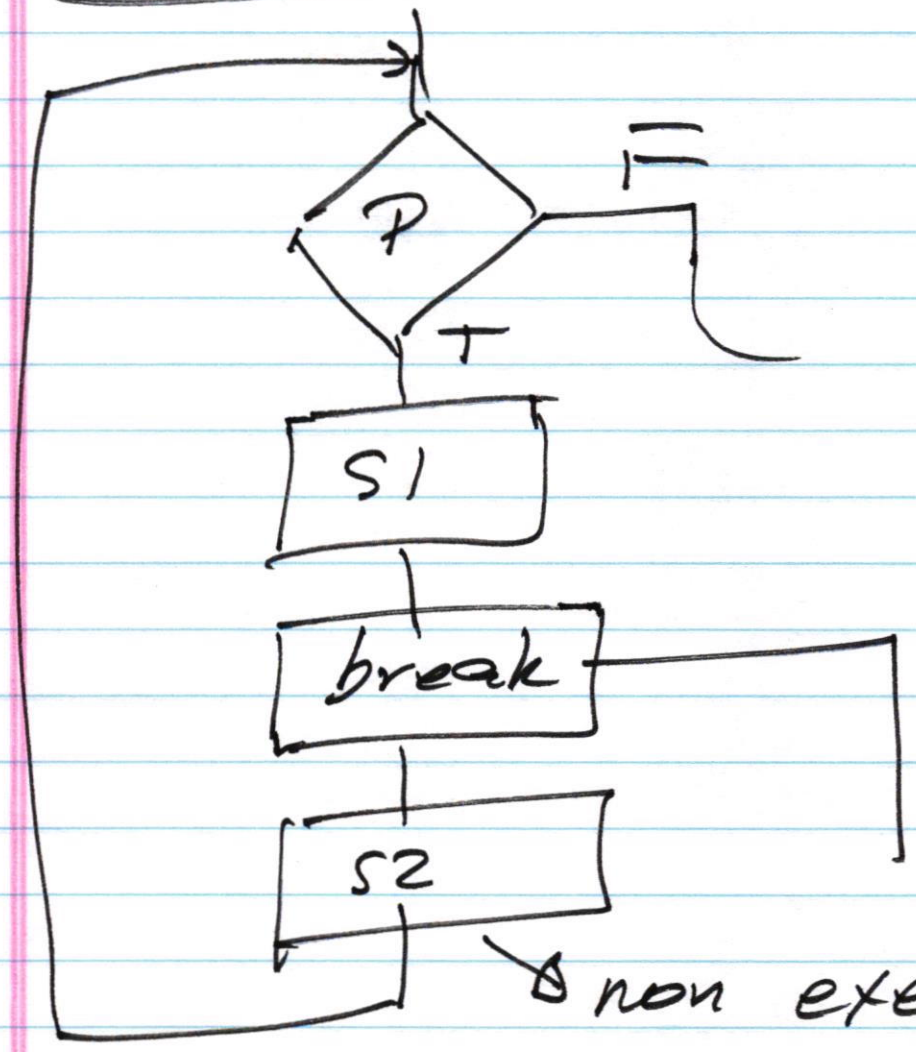
coverage

100%

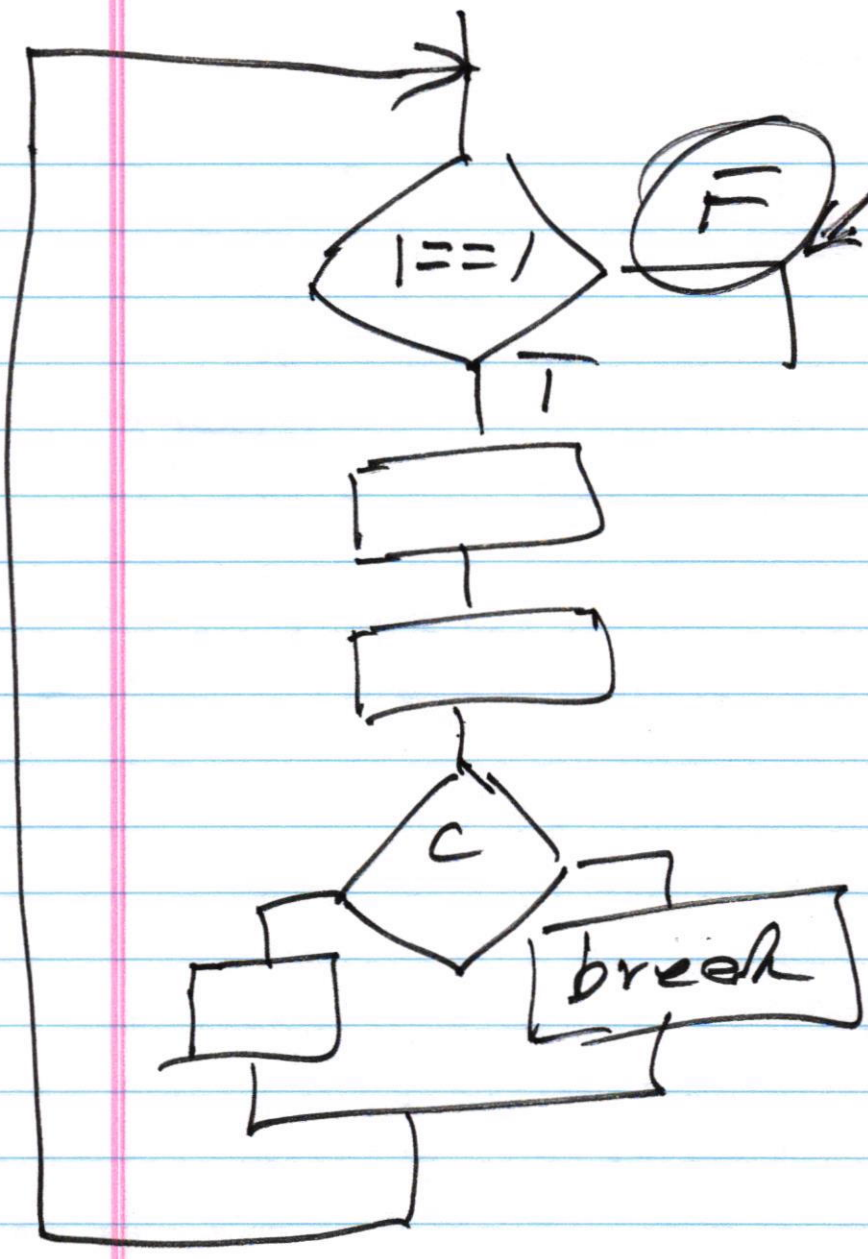
* branch testing is more demanding than statement testing.

* when branch testing is satisfied (100% coverage) the statement testing is also satisfied.

non-executable statements / branches



error in source code!!!



non-executable branch

defect??

this may be ok!?

Homework #1

17 conditions

Problem #3

2^{17}

conditions
actions.

$\Downarrow$

use "don't care"

C1: $0 \leq E1 \leq 100$
C2: $0 \leq E2 \leq 100$
C3 $0 \leq Pr. \leq 100$
C4: Student#: 9 char.
C5: format A XXXXXXXXX
C6: $0 \leq E1 < 50$
C7: $50 \leq E1 < 60$
C8 $60 \leq E1 \leq 100$

E2
⋮
Pr

Cn $\frac{e1+e2+pr}{3} \geq 75$

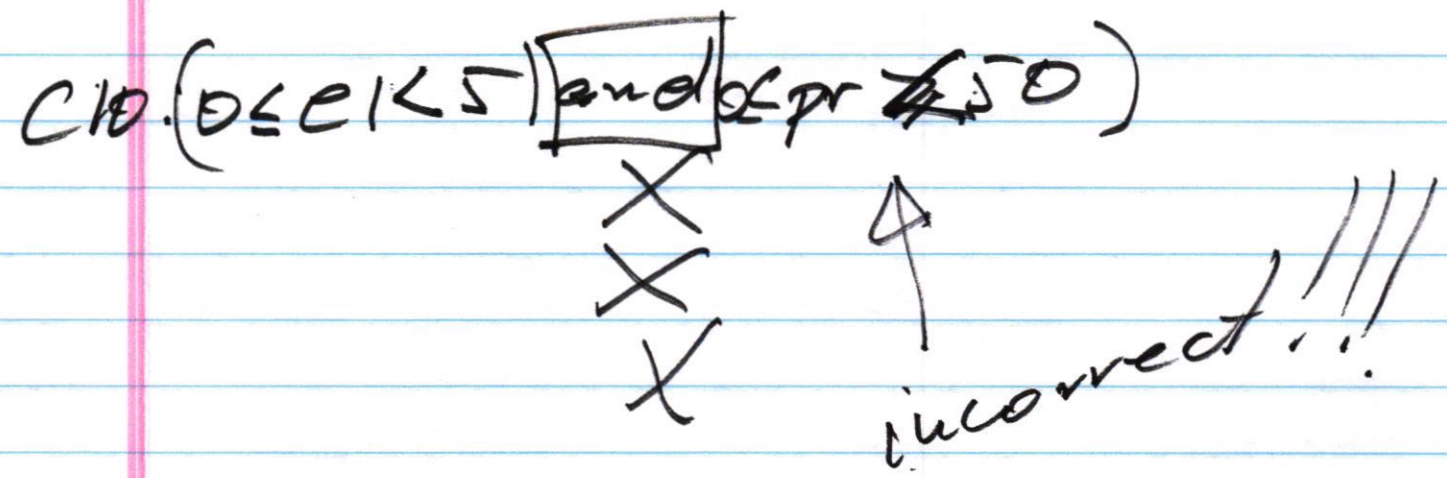
---

a1: A
a2 B
a3 C
a4 E

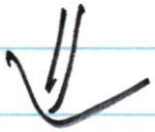
a5: invalid input
a6: impossible.

Conditions should be simple

No: "and", "or"



create decision table



a set of rules

test case

one test per a rule

you do not have to create a test for "impossible" combinations of conditions

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-1<50) and (Exam-2 ≥ 60)

is **not acceptable** as a condition in the decision table. However, "*Exam-1 <50*" is a condition; "*Exam-2 ≥ 60*" is a condition.