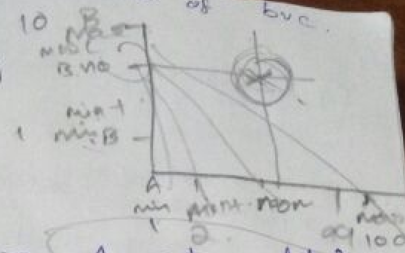
Robustness testing

* upgrade version of bvc.

No of TC $= (6 * n) + 1$

(before creating TC table) $= 6 * 2 + 1$

$= 13$ TC



* (9-13) = 4 here other 4 are Amax+ and Amin-
Bmax+ and Bmin-

TCID	Test I/P	Expected Result	Actual Result	Status
10.	Amin- (0)	Bnom (25)	5	
11	Amax+ (101)	Bnom (5)	1.6	
12	B min- 50	Bmin- (0)	5.0	
13	A nom 50	Bmax+ (11)	61	

worstcase Testing:-

No of TC $= (5^n)$

$= 5^2 = 25$ test cases

* only for limited no of data input it is tested.

* derived version of BV checking

* so already 9 TC created (BVC)
$9 - 25 = 16$ - TC needs to created

written in previous scan

BVC :-

No of testcase = $(4 * n) + 1$ (n = no of data)

$= (4 \times 2) + 1$

$= 8 + 1 = 9$.

TC_ID	Test I/P A (1)	Test I/P B (5)	Expected Result	Actual Result	Status
1	Amin	Bnom	6	6	P
2	Amin+ (1)	Bnom (5)	7	8	F
3	Amax- (99)	Bnom (5)	109	0	F
4	Amax (100)	Bnom (5)	105	105	P.
5	Anom (50)	Bnom (5)	55		
6	Anom	Bmin	51		
7	Anom	Bmin+	52		
8	Anom	Bmax-	59		
9.	Anom	Bmax	60.		
10					
11					
12					
13					
14					
15					
16					

* using this every tester has unique testcase fixed no of testcase.

Amin

Amin+

Amax-

Amax

Equivalence class partitioning
(Ecp. testing)

* minimize testcase than BVA
* Remove duplicate testcases

* Input is equivalaly divided into
  - ↳ valid input.
  - ↳ invalid input.

Rules :-

1. Range of values
   → ex: int x (1-100) { if. take any three testcase.

if we use BVC minimum 5 TC require

Amin
Bmax
nom
min -
max -.

condition:-

1). x = b/w (1-100)
↳ x value between the range.

2) x = x < 1 x value must be lesser than (lower limit)

3). x = x > 100. x value must be greater than higher limit

1: valid
2 and 3: invalid.

Rule2: "Number of values"

* Specify number.
* two testcase enough.
  - valid. (eg Reg 377 = 220 credits)
  - invalid. (eg Reg 377 ≠ 220 credits)

* For invalid asume the valid value and take opposite of it

Rule3: "must be"

* They specify anything apart from Number.
* Two testcase:
  - valid
  - invalid.

valid: username must be lowercase.
Invalid: username must not be lowercase.

Steps for ECP

1. List out no of conditions and Inputs.
2. map each conditions against ECP rules
3. Create ECP - table
4. create testcase table with the inclusion of equivalent classes.
5. Remove duplicate testcases.

Pg) as a tester test the password field gmail account is getting the input as 3 → 15 alphanumeric char of which 1st two char must be alphabet :-

step1

conditions :

1. Pwd 3-15 range
2. Combination of alphanumeric
3. 1st two char must be alphabet

step2

map each condition against ecp rule.

↳ cond 1 ← Range of values

valid (rc) 1. pwd = b/w 3-15.

Invalid 2. pwd = pwd < 3 char.

Invalid. 3. pwd = pwd > 15 char.

Rename this label. → EC 1.

→ EC 2.

→ EC 3.

↳ cond 2 : combination of alphanumeric

valid — (rc) 1. pwd. = alphanumeric.

Invalid - 2. pwd ≠ alphanumeric.

→ Ec 4.

→ EC 5 (rename label here cont... process)

4 con 3 :

first 2 char must be alphanumeric.

Ec6 <= valid → pwd = 1st two char must be alpha

Ec7. <= Invalid → pwd : 1st two char must not be alpha.

or

≠ 1st two char must be alpha.

step3

create Ecp table.

condition	valid Ec.	Invalid Ec.
1	Ec1	Ec2, Ec3.
2	Ec4	Ec5
3	Ec6	Ec7.

step4

Test case table.

No of TC = No of Ec's

= 7 Testcase.

TC ID	Test ip pwd.	EP	AR.	S.	Ec - coverage
1	ab123	accept.	accept	pass.	Ec1, EcA, Ec6
2	ab	not accept.			Ec2 → Ec5
3	1234...20	not accept.			Ec3, Ec7
4	xyz22	accept.			EcA.
5	$	not accept.			Ec5
6	ab12.	accept.			Ec6
7	12ab	not accept.			Ec7

* 1st TC cover Ec1 go to 1st condition.
  Check valid conditions.
  here: Ec1 (ab123)
  EcA ↙ satisfied
  Ec6 (all valid)

* 2nd TC. Correlate Ec2.
  ↳ Invalid (never bother other).
  (Less than 3 char) ab.

* 3rd TC. Ec3 – Invalid.

* 4th TC. Ec4 – If valid consider all valids but it covers all valids.

note: if invalid (Ec2) take only (Ec2) don't consider.
if valid (Ec1) take other valids (EcA, Ec6)
and input obey others valid.
F/P = (ab123)

after fill all TTC

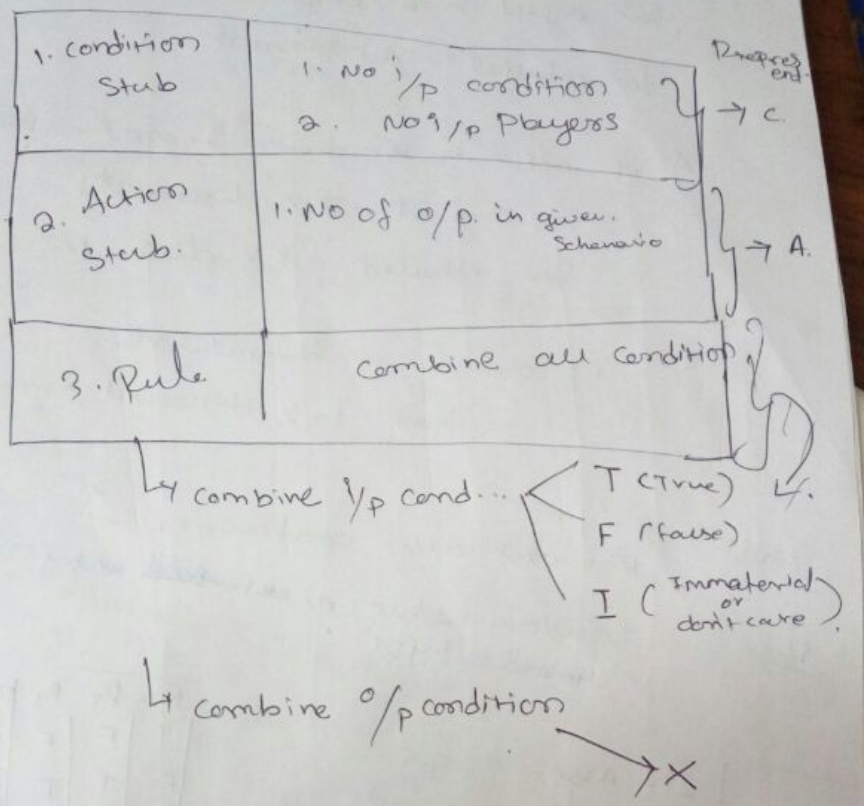
If 1st covers all EC4, EC6
then strict.

the same. I/p covers EC7 (1st two char not alpha)

Final table. testcase -

	Ipf	ER	AR	S	Ec - coverage.
1.	ab123				EC1, EC4, EC6
2.	ab.				EC2, EC5
3.	1231				EC3, EC7.

Decision table. → condition table.
template.



Eg A shop owner is having some goods to save. different types of customers with different discounts rates.

1. If order is given by ABC company irrespective to order value. gives flat 10% discount rate.

2. if Order val > Rs. 50,000/- for agent give 15% discount
for retaler - 12% discount rate

3. if order val 20000 and less than 50000/-

for agent - 12% Discount

for retailer - 10% Discount

4. if order is less than 20000/-

for agent - 8% discount.

for retailer - 5% discount.

5. if order is furniture.

flat 10% discount irrespective

Step1: list out all condition

Step2: Decision Table (or) extended entry (DT).
(or) limited entry (DT).

		R1	R2	R3	R4	R5	R6	R7	R8
Condition Stub	C1. ABC	T	F	F	F	F	F	F	I
	C2: agent	F	T	F	T	F	T	F	I
	C3: retailer	F	F	T	F	T	F	T	I
	C4: O > 50000/-	I	T	T	T	T	F	F	I
	C5: 20000 < O < 50000/-	I	F	F	F	F	F	F	I
	C6. O < 20000/-	I	F	F	F	F	F	F	I
	C7. O = furniture	T	F	F	F	F	F	F	T
Action stub	A1. 10% D.R.	X	-	-	-	X	-	-	X
	A2. 15% DR.	-	X	-	-	-	-	-	-
	A3. 12% DR.	-	-	X	X	-	-	-	-
	A4. 8% DR	-	-	-	-	-	X	-	-
	A5. 5% DR.	-	-	-	-	-	-	X	-

(R1 - order company must be ABC so T other players (F)
and value order (I don't care)
so put I)
expected o/p is 10%
so put X

3. T.C. table : = No of rules in d.c table.
= 8 (T.C.)

Give your own.

TC ID	Test i/p			EP	AR	Status.
	type of cust	Ord value	Furnit?			
1	ABC	I could not understand 50000	NOT	10%.	10%.	pass
2	Agent	55000	NOT	15%.	15%.	pass.
3	~~Retailer~~ agent	~~35000~~ 21000.	NOT	12%.	13%.	fail
4	agent	21000.	NOT	12%.	12%.	pass.
5	Retailer	30000	NOT	10%.		
6	Agent	70000	Not	8%.		
7	retail	15000	NOT	5%.		
8	agent ABC or	100001	yes	10%.		

TC ID mapped to Rules.

TC ID - see rule 1 and true is ABC and order value don't care so this amount and furniture is Null.

Rule 2 -

4. TC

No of TC = No of rules

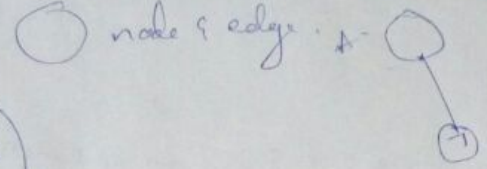
= 5

TC ID	Test I/P		E.R.	A.R	Status
	1st char	2nd char			
TC 1	A	5	valid.	valid	pass.
TC 2	B	1	valid.	valid	pass
TC 3	C (not A and not B)	7	invalid	invalid.	pass
TC A	A	Z (not num)	wrong	valid.	fail
TC J	B	$ (not num)	wrong	invalid	fail

pg with read only (true char)

user value

step 2



Final A → A

Steps Cyclomatic complexity (notation of graphical)

1. $V(G) = (e - n) + 2 * P$

e - edges

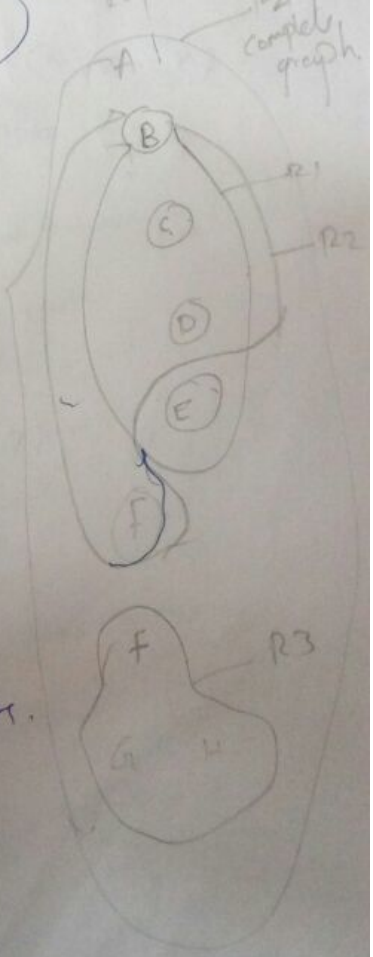
n - nodes

P: ~~no of decision~~ no of Function block.

2. $V(G) = d + 1$

d: ~~no~~ decision

3. V(G) = no (region) = n(2)



no of edges = 10

formula 1) $V(G) = (e-n) + 2 * P$ — no of function blocks (main c)

$= 10 - 8 + 2 \times 1$

$= 4$

f2) $V(G) = d+1$

$= d+1 = 3+1 = 4$

f3) $V(G) = n(R)$

Region based on D.D graph.

$= n(R_1 + R_2 + R_3 + R_4)$

$= 4$.

If $(f1 = f2 = f3)$

if all same formula values same : finite loops prog has.

if any one not match.
infinite loops

output values should cover path.

= P should cover path

4) Independent paths A → H.

1. A - B - C - D - f - G - H. ✓
2. A - B - C - D - f - H ✓
3. A - B - f - G - H. ✓
4. A - B - f - H. X - not have G and D
5. A - B - C - E - B - f - G - H ✓
6. A - B - C - E - B - f - H X
7. A - B - C - E - B - C - D - f - G - H. X
8. A - B - C - E - B - C - D - f - H. ✓

Note

path _ has D and G - it print two printoutput.

only 4 valid path or independent path

(2, 3, 5, 8) ✓

Step 5:-

No of. TC = No of valid paths

= 4.

TC NO.	Test I/p	ER	AR	Status	paths covering.
1	4	not prime	NP	pass	A B C D f H
2	2.	prime.	prime	pass	A B f G H
3	7	prime	Prime	pass	A B f E B f G H.
4	6	not prime	not prime	pass	A B C E B C D f H.

Work:

State	Line no:	Anomaly
~d	5	Normal
dv	5-6	Normal
vv	6-8, 8-10, 10-11, 11-14	Normal
v~	14-21	Normal
k~	21	Normal

Payment:

State	Line No	Anomaly
~d	4	Normal
DD	4-7	Normal
DV	7-11	Normal
dv	11-14	Normal
dv	14-15	Normal
44	15-16	Normal
dv	16-20	Normal
k~	21	Normal

Control flow

(i) define

(ii) usage < P-use
C-use

(iii) kill

(i) define: 4, 7, 11, 14, 16

(ii) usage:

C-use: 11, 14, 16

P-use: 15

Usage: 20

(iii) kill: 21

(*) (Mention everything in graph) in exam.

Step 3:

anomaly table:

n variables : n - anomaly table.

variables:

work

Payment.

15- if (paymoul >= 3000)

16- paymoul = paymoul * 0.9;

17- }

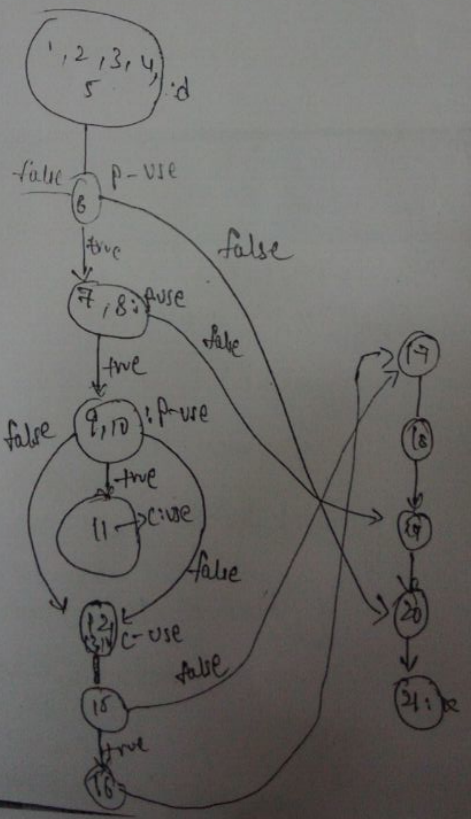
18- }

19- }

20- printf("final paymoul", paymenth);

21- }

Step 2:



Ex: DD.

Term Ex: a = 10;
b = 22;

Steps:

1. put line number for every statements in your program.
2. Draw control flow graph.
3. Create anomaly table.

Step 1:

```
1- main()
2- {
3-   int work;
4-   double payment = 0;
5-   scanf("%d", &work);
6-   if (work > 0) {
7-     payment = 40;
8-     if (work > 20)
9-     {
10-      if (work <= 30)
11-       payment = payment + (work - 25) * 0.5;
12-     else
13-       payment = payment + 50 + (work - 30) *
14-
```

Anomaly table → Single character
→ Two "

State	meaning	Anomaly
~d	first define	Normally
~u	first use	serious bug
~k	first kill	potential bug
d~	def last	Serious Bug
u~	use "	Normal
k~	kill "	Normal

Two-character anomaly table.

State	meaning	Anomaly
dd	def-def	minor bug
du	def-use	Normal.
dk	def-kill	potential bug
ud	use-def	serious bug
uu	use-use	Normal
uk	use-kill	Normal
kd	kill-def	serious bug
ku	kill-use	serious.
kk	kill-kill	minor bug.

(i) define state: The definition of data in the program i.e., in which line number we get the value for the data.

Ex:
```
int main() {
    int a, b;
    float c;
cin>>a (or) a=10;   --> define state
    b = a + 5       --> usage state (c-use)
}
```

(ii) usage state: In which line number the value of the data is used.

It has two classifications:

1. c-use (computational use);
2. p-use (predicate use)

```
b = a + 5
if(a > 5)
```

(iii) Kill(x): The memory allocated for the data will be deleted here.

Ex: free(a)

24/6/16
Thursday

(*) Data flow testing : Testing the flow of data (variables) in the pgm by the developer

Last

↓
(variable in the pgm)

Terminology on state.

To test the flow of data we use 3 terminologies :

① Define (d)
② Usage (u)
③ kill (k)

6. A-B-C-E-B-F-H ✗

7. A-B-C-E-B-C-D-F-G-H ✗

8. A-B-C-E-B-C-D-F-H ✓

(5) Check the valid paths based on the o/p of the program.

Test Case Tables

No. of TC = No. of valid paths.

= 4

TC_ID	Test i/p	ER	AR	Status	Independent valid path coverage
1	4	Not prime	prime	fail	A B C D F H
2	2	prime	prime	pass	A B F G H
3	3	prime	Prime	P	A B C E B F G H
4	6	Not prime	Not prime	Pass	A B C E B C D F H

3. $V(G) = n(R)$

$R_1 = (B - C - E - B)$

$R_2 = (B - C - D - F - B)$

$R_3 = F - G - H$

R_4 - Complete graph.

$\therefore V(G) = n(R)$

$= n(R_1 + R_2 + R_3 + R_4)$

$= 4$

① ≠ ② ≠ ③ - finite.

else Infinite.

Step 4: Independent paths: $A \rightarrow H$

1. A - B - C - D - F - G - H ✗
2. A - B - C - D - F - H ✓
3. A - B - F - G - H ✓
4. A - B - F - H ✗
5. A - B - C - E - B - F - G - H ✓

Step 3: Calculate Cyclomatic complexity.
Now find the flow

1. $V(G) = e - n + 2p$

↓

$V(G) - C \cdot C$

e - edges

n - nodes

p - no: of function blocks.

2) $V(G) = d + 1$

d - No: of decision (or) conditional statements

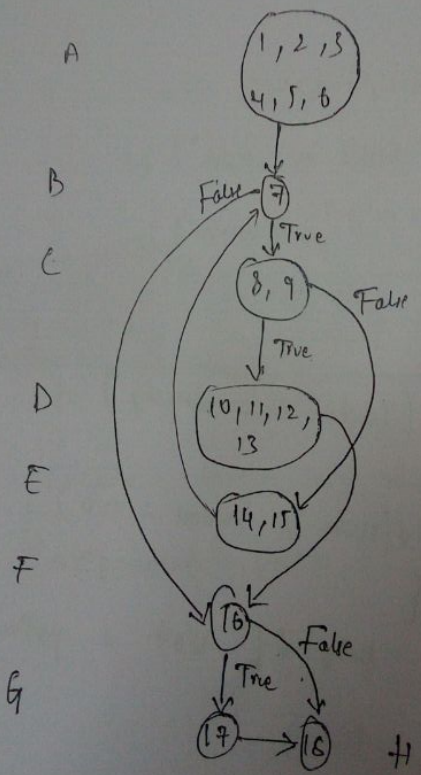
3. $V(G) =$ no of regions
$= n(R)$

So, according to program.

1. $V(G) = 10 - 8 + 2 \times 1$
$= 4$

2. $V(G) = d + 1$
$= 3 + 1 = 4$

```
15 - }
16 - if (index == num)
17 -     printf("prime num");
18 - }
```



Path Testing

Wednesday

Steps : Draw D-D Graph

```
1- main()
2- {
3-  int num, index;
4-  printf("Enter a num");
5-  scanf("%d", &num);
6-  index = 2;
7-  while (index <= num-1)
8-  { if(num % index == 0)
9-  {
10-  printf("not a prime");
11-  break;
12-  }
13-  index++;
14-
```

6/18
Wednes

Path testing

Steps: Draw D-D Graph

```
1- main()
2- {
3-   int num, index;
4-   printf("Enter a num");
5-   scanf("%d", &num);
6-   index = 2;
7-   while (index <= num-1)
8-   {  if(num % index == 0)
9-   {
10-    printf("not a prime");
11-
12-    break;
13-  }
14-  index++;
```