★ <u>Component Testing</u>=>

↳ software components are often composite components made up of several interacting objects.

↳ component interface is used to access the functionality of objects.

  ↳ If linked objects cannot access the component, the interface is wrong.

↳ Testing focuses on showing component interface behaves according to its specification.

  [assumption that each component works solely well]

★ <u>Interface Testing</u>=>

↳ objectives are to detect faults due to interface errors or invalid assumptions.

↳ <u>Parameter interfaces</u>=> Data passed from one method to other.

↳ <u>SHM interface</u>=> Block of memory shared b/w functions.

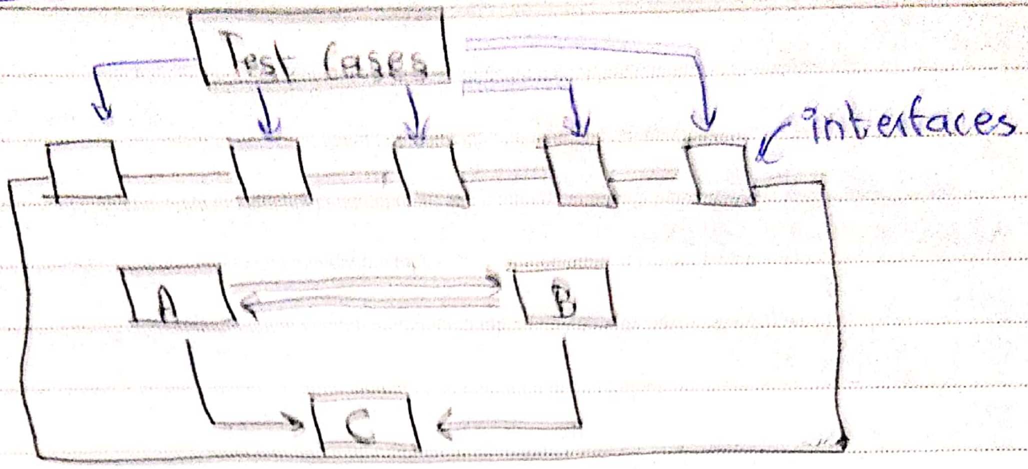
↳ <u>Procedural interface</u>=> sub-system encapsulates procedures to be called by other subsystems. Unki testing

↳ <u>Message Passing</u>=> subsystem req. service from others.

  => MPI

Fault propagation

Dated:



↳ Interface errors ⇒

↳ interface misuse ⇒ a calling component makes a mistake in use of the callee's interface.
[parameters in wrong order] [a calling c bf4 b]

↳ interface misunderstanding ⇒ assuming the wrong behavior of a component.
[binary search called with unsorted array]

↳ timing errors ⇒ components operating at different speeds access out-of-date information. [Race conditions]

↳ guidelines ⇒

↳ design tests so parameters to a called procedure are at the extreme ends of their ranges.

↳ testing segmentation faults [null ptr]

↳ design corner test cases [comp. fail]

↳ use stress testing [message passing] [timing errors]

↳ In SHM, vary the order of access.

Scanned with CamScanner

Dated: ____________

★ System Testing ⇒
↳ integrate components to create a version of the system and test the integrated system.
↳ testing component interactions is the goal
↳ checks that components are compatible, interact correctly & transfer right data at right time.

★ Use-case testing ⇒
↳ identify system interactions.
↳ each use case involves several components so they force the interactions to occur.
↳ sequence diagram documents these interactions.

★ Test Driven Development ⇒
↳ interleave testing & development.
↳ Tests are written before code and passing the test is necessary.
↳ code dev is incremental, along with a test for that increment. Can't move on before passing all tests.
↳ Benefits ⇒
  ↳ code coverage
  ↳ regression testing [check if new func is causing prob to prev]
  ↳ simplified debugging
  ↳ easy documentation

Cyclomatic complexity => minimize test cases

Dated:

★ **Basis Path Testing** => White box Testing

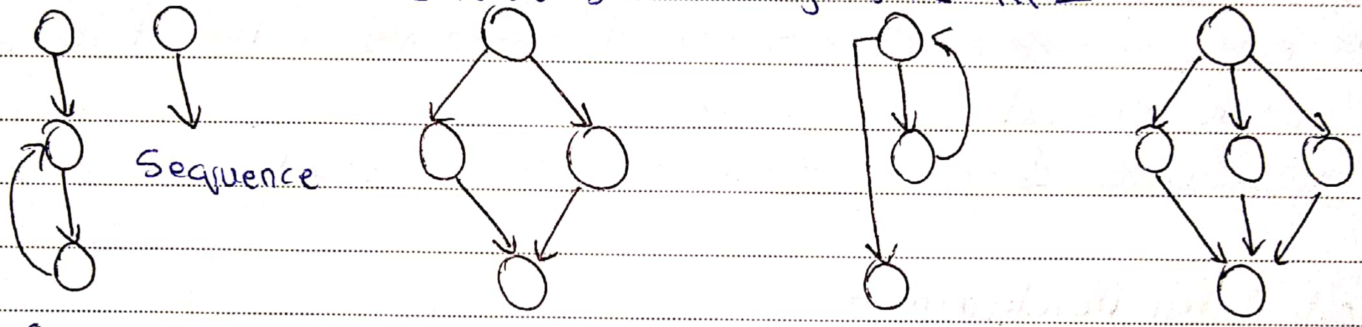
1) Draw the control flow graph.
2) Calculate Cyclomatic complexity using all methods.
3) List all Independent Paths.
4) Design test cases from independent paths.

~~& Predicate Nodes~~

$CC = E - N + 2P$ Connected components

→ ~~Dividing into two paths~~

→ can also be done with bounded regions i.e : R+1

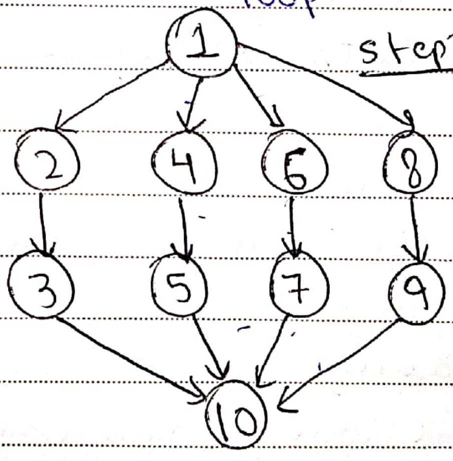


for loop   if-else   loop   switch

Step1

1) Input a,b,c
2) If
3) output
4) else if
5) output
6) elseif
7) output
8) else ~~if~~
9) output
10) return

step2



E = 12

N = 10

P = 1

$CC = E - N + 2P$

$CC = 12 - 10 + 2$

$\boxed{CC = 4}$

step3

Paths => 1-2-3-10

1-4-5-10

1-6-7-10

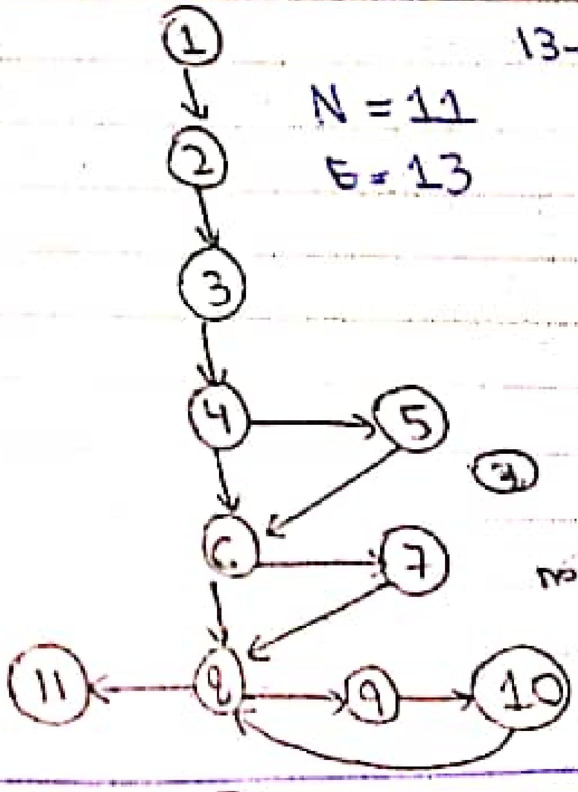
1-8-9-10

Dated: ________

Step 4 create Tests

if	else if	else if	else	Expected Outcome
T	F	F	F	—
F	T	F	F	—
F	F	T	F	—
F	F	F	T	—

1) a=10
2) input
3) input
4) if
5) out
6) if
7) out
8) while
9) do
10) do
11) end

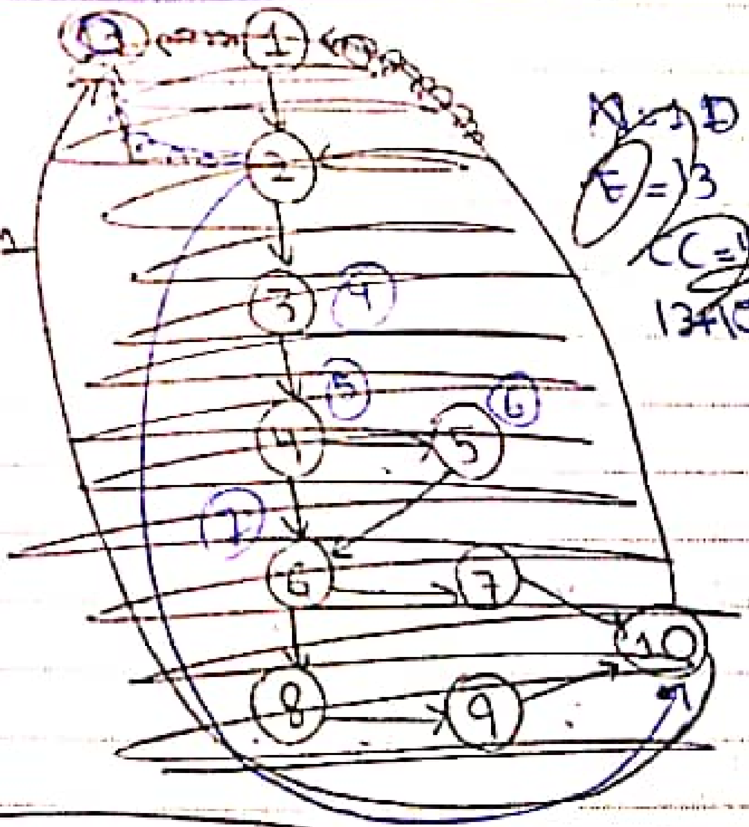


13-11+2

N = 11
E = 13

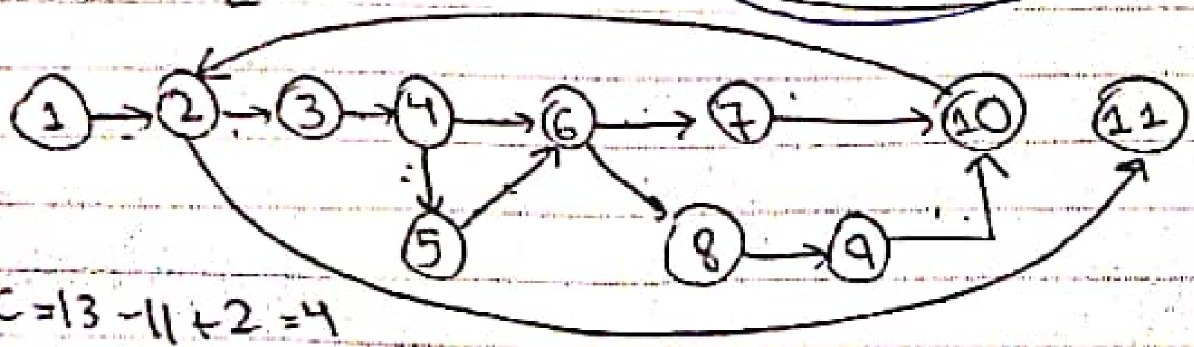
$\boxed{CC=4}$

1) values
2) while
3) assign
4) if
5) do
6) if
7) do
8) else
9) do
10) end
11) ~~[illegible] end~~



~~N=10~~
~~E=13~~
~~CC=4~~
~~13-10+2=5~~

N=11
E=13



CC=13-11+2=4