Vrije Universiteit, Department of Computer Science

Examination paper for **Software Testing -SOLUTIONS**
31 March 2017 12:00-14:45 MF-FG2

This is a closed book written exam.

No printed material or electronic devices are admitted for use during the exam.

The answers have to be given in English.

Both homework and exam are compulsory and graded on a 1 to 10 scale.

The exam grade is calculated as ((Q1+Q2+Q3+Q4+Q5)*0.9/0.8 +10)/10.

The final grade is calculated as 0.6*homework + 0.4*exam

A pass is given only if both homework and exam components are >= 5.5

	Q1 (concepts)	**Q2**	**Q3**	**Q4**	**Q5 (code)**	**Σ Qi**	**Maximum credits**
a)	**3**	**9**	**9**	**9**	**3**		
b)	**3**				**5**		
c)	**3**				**3**		
d)	**3**				**3**		
e)	**7**				**7**		
f)	**7**				**3**		
g)					**3**		
Total	**26**	**9**	**9**	**9**	**27**	**80**	**10**

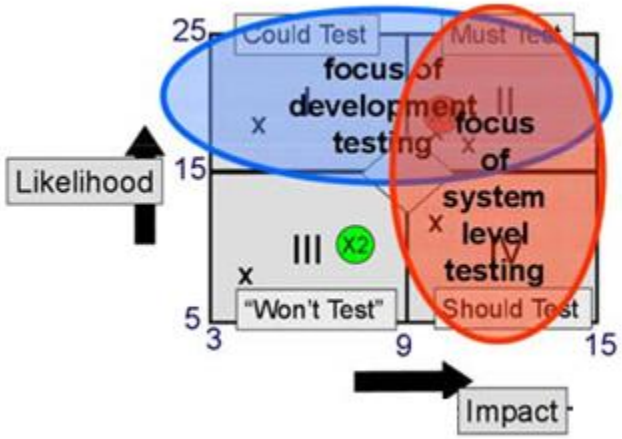
Q1. Concepts [26p]

a. Define the terms ON and OFF point. Give an example. [3p]
b. What is the difference between a verification of a SRS and its validation? [3p]
c. Define the terms safety and reliability and show that they don't mean always the same. [3p]
d. Define the term Product Risk Matrix and explain its role in testing. [3p]
e. Consider the heater control system built for a neonatal incubator, a rigid box-like enclosure in which a premature born baby can be kept in a controlled environment, for observation and care. The goal is to keep the baby's temperature at 37°C ± 0.1 °C. Apply the first step of STAMP analysis to this heating control system. Identify one possible accident. Draw a simple control structure for this system. Identify based on this control structure some unsafe control actions. [7p]
f. Explain what symbolic execution is and show how it can be used to generate test inputs for this program: [7p]

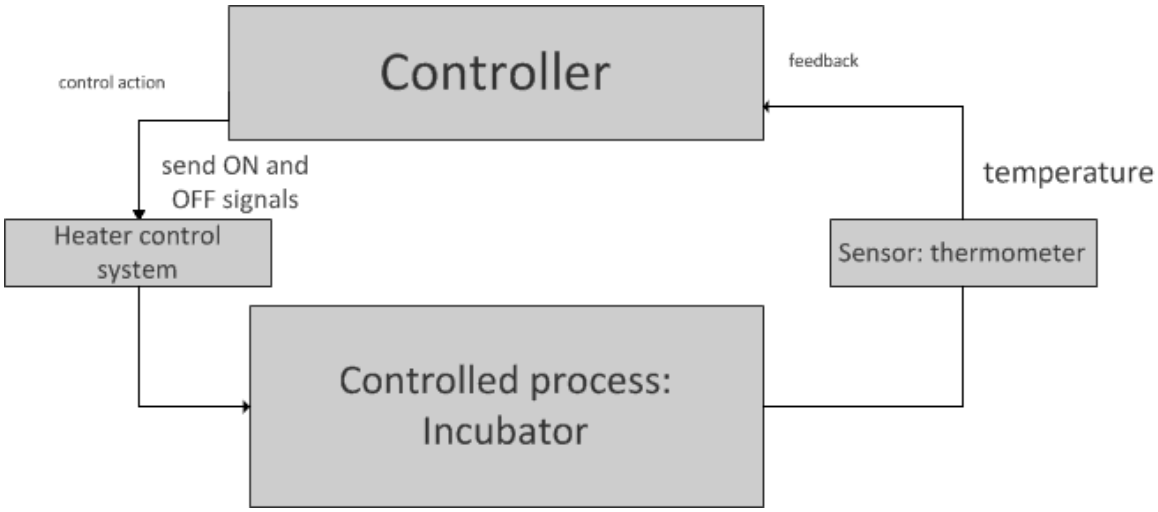
```
   int x, y;
1  if(x > y){
2     x = x+y;
3     y = x-y;
4     x = x-y;
5     if(x - y > 0)
6        assert false;
7  }
8  print(x, y)
```

a) these are concepts used in domain testing . An ON points is always on a boundary, an OFF point is close to the boundary, but outside the domain if the boundary is closed and inside the domain if the boundary is open (COOOOI rule). Examples, for x<10, an ON point is x=10 and an OFF point is 9 (open boundary, inside the domain).
b) Verification of a SRS checks this document for properties such as complete, clear, consistent, testable, not ambiguous, etc and validation means to ask the user if this is what he really wishes.
c) Safety is freedom of accidents. Reliability is the probability that a system will function as expected in a period of time and in a certain environment. They don't mean the same. A reliable system may be not safe – hairdryer in a bathtub, and for example an emergency shut down of a nuclear plant that do not follow the legal procedures is safe but not reliable.
d) a product risk matrix is a graph that has risk likehood on one axis and the impact on the other. All functionalities get a point in this graph. The matrix It helps to prioritize testing under time pressure.

Control action	The control action is not given	An incorrect control action is given	The control action is given at the wrong time or wrong order	The control action is stopped too soon or applied too long
Send On and OFF signals	No ON command is given to heater when the temp is lower than 36.9C No OFF command is given when the temperature is higher than 37.1C.	Heater is turned ON when the temperature is already 37.1 C and rising Heater is turned OFF when temperature is 36.9C.	Heater is turned ON long after it was necessary Heater is turned OFF long after it was necessary	



e) Accident: Baby dies because overheating or underheating.

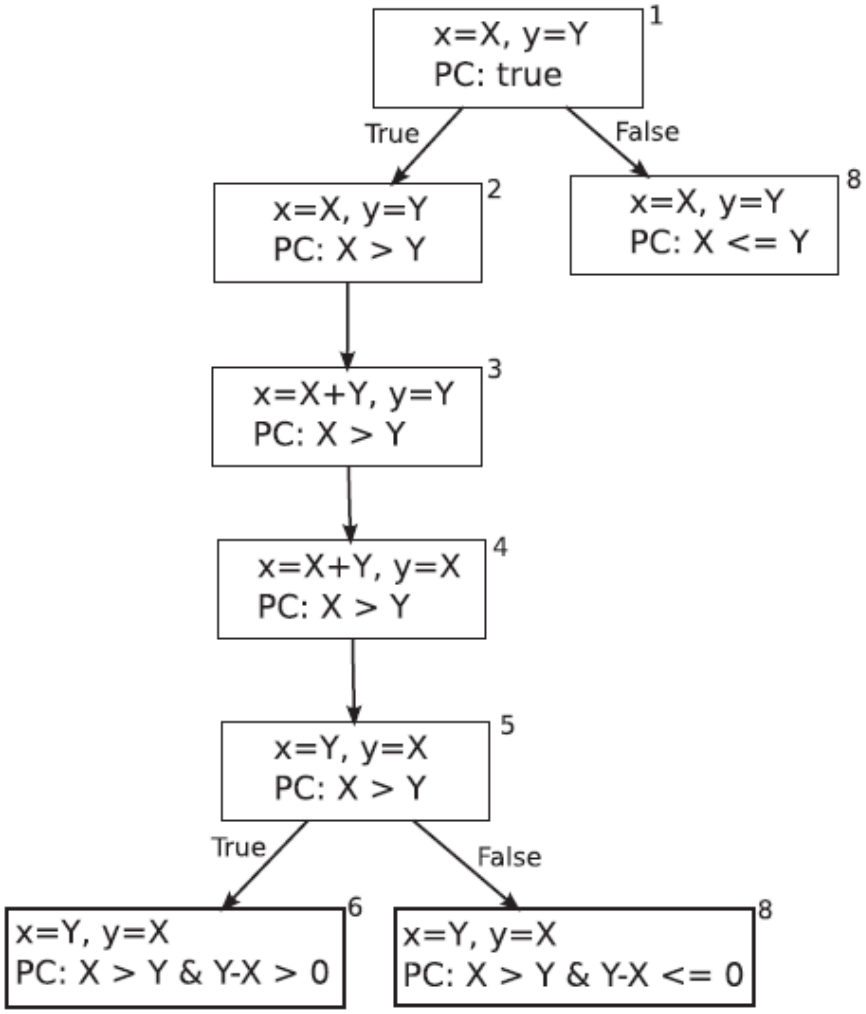


The control action is Send ON and OFF signals. For this control action we build a table for unsafe control actions.

f).

Symbolic execution is a program analysis technique that executes programs with symbolic rather than concrete inputs

It maintains a path condition or constraint (PC) that is updated whenever a branch instruction is executed, to encode the constraints on the inputs that reach that program point. The flow graph and a possible set of test cases are shown below.



Path	PC	Program Input
1,8	X <= Y	X=1 Y=1
1,2,3,4,5,8	X>Y & Y-X<=0	X=2 Y=1
1,2,3,4,5,6	X>Y & Y-X>0	none

Testing from requirements Q2-Q4 [27p]

Q2. [9p]

Consider the following requirement:

`[FR 1] The system shall allow shipments for which the price is less than or equal to €200.`

Design and generate test cases to defensively test this requirement, by using equivalence partitioning (EP) and boundary value analysis (BVA). Justify your test cases specifications and minimize your test cases.

First we need to make some assumptions. The price is >=0. the increment is 1 euro.

EP will give us a few equivalence classes:
EP1: price < 0 invalid class IC
EP2: price >= 0 and price <=200. valid class VC
EP3: price > 200. invalid class IC

EP4: not a number : IC
EP5: empty string : IC

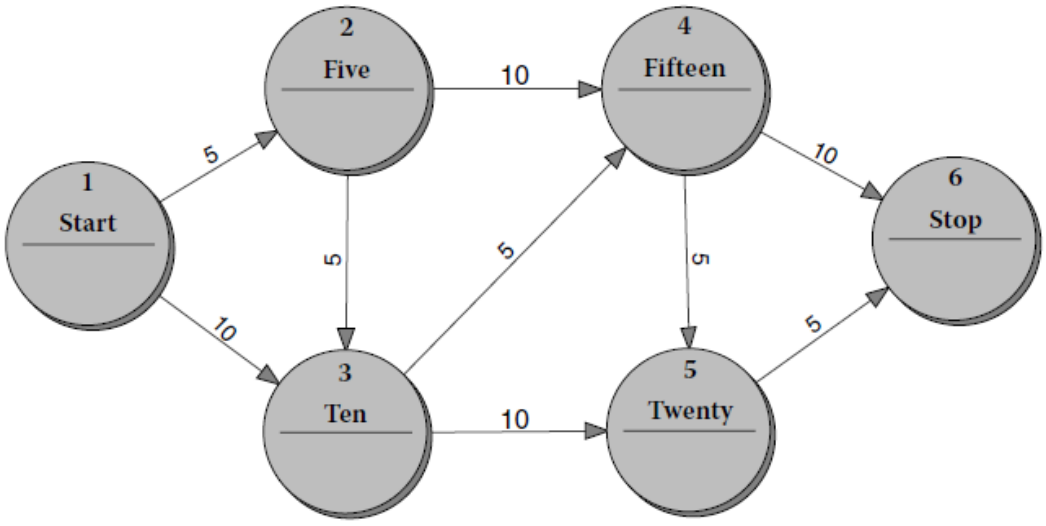
BVA will add some test cases such as min, min+1, nom, max-1, max. This is a reasonable set of test cases:

ID	Input	specification	expected output
TC1	-50	EP1	not allowed
TC2	-1	EP1+BVA	not allowed
TC3	0	EP1+BVA	allowed
TC4	1	EP2+BVA	allowed
TC5	100	EP2+BVA	allowed
TC6	199	EP2+BVA	allowed
TC7	200	EP2+BVA	allowed
TC8	201	EP3+BVA	not allowed
TC9	500	EP3	not allowed
TC10	QWERT	EP4	not allowed
TC11	" "	EP5	not allowed

Q3. [9p]

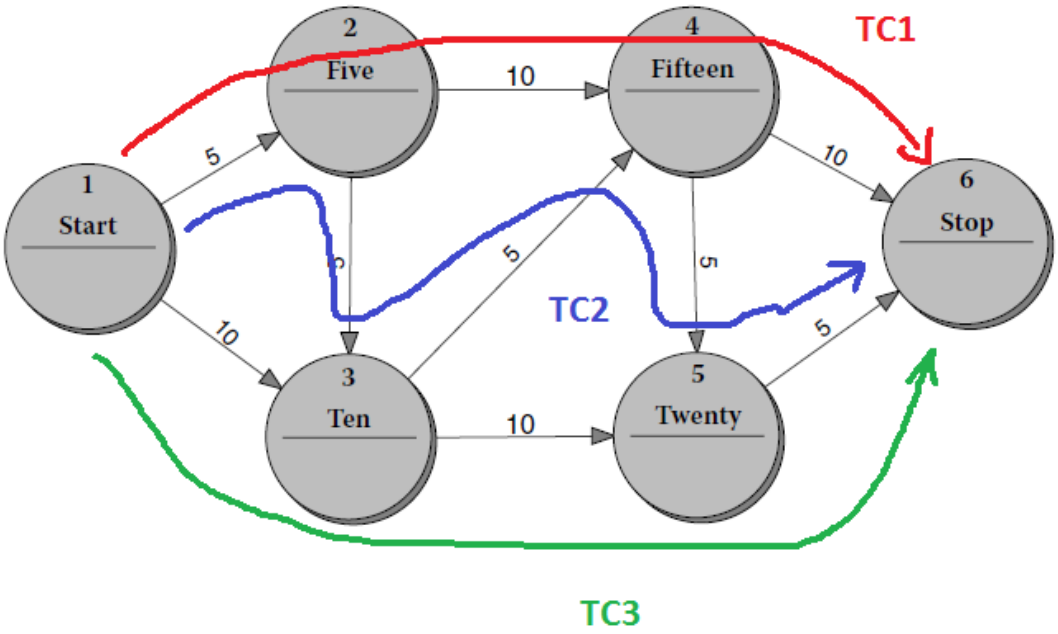
A control system has to count the amount of money dropped into a vending machine. Only 5 and 10 cent coins are accepted. The correct, recognized sum is 25 cents.

a) Model this system with a state transition diagram.
b) Generate test cases from this diagram. Argument your approach.



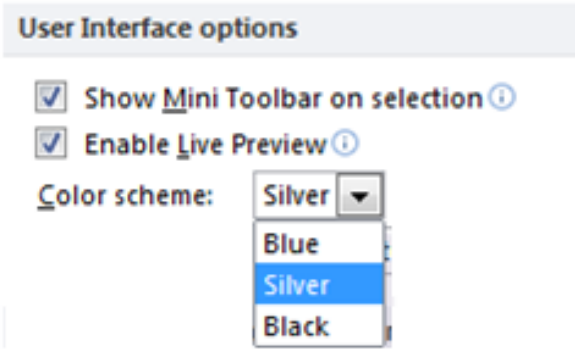
We chose for 100% transition coverage and we obtain the following test cases.

TC1. Start state: Start. Input 5 cents, 10 cents, 10 cents. Expected state Stop.
TC2. Start state: start. Input 5-5-5-5-5. Expected state: Stop.
TC3. Start state: Start. Input 10-10-5. Expected state : Stop.



Q4. [9p]

Given the preferences menu shown below.



a) Identify the inputs. How many test cases do you need to exhaustively test this menu?
b) Design test cases using a pairwise combinatorial model. You should use a suitable orthogonal array from the list in the appendix.

a) we have the following inputs (factors)
show minitool: with 2 values (levels)
enable live preview: with 2 values
color scheme. with 3 possible values:
In total we need 2 x 2 x 3 = 12 possible inputs for exhaustive testing.
b) we take the 3^ 4 array, that uses 4 inputs that can have 3 possible values. We don't use it all. We use only the first 3 columns because we have only 3 inputs. We map the 3 columns to the 3 inputs.

SEC
000
012
021
102
111
120
201
210
222

S and E do not need the value 2. Instead we use one of the two possible values, true of false. We map the problem to the OA as follows:
Show: 0 hidden, 1 visible, 2 does not care
Enable: 0 enable 1 disable 2 does not care
Color scheme: 0- Blue 1-Silver 2- Black

SEC
012
0x1
102
111
1x0
x01
x10
xx2

SEC
012
011
102
111
100

101
010
~~012~~

Depending on how we fill this x, we can discard the redundant lines.
A possible test suite is:

ID	Show	Enable	Colour scheme
TC1	0	1	2 Black
TC2	0	1	1 Blue
TC3	1	0	2 Black
TC4	1	1	1 Silver
TC5	1	0	0 Blue
TC6	1	0	1 Silver
TC7	0	1	0 Blue

Q5. Code based testing [27p]

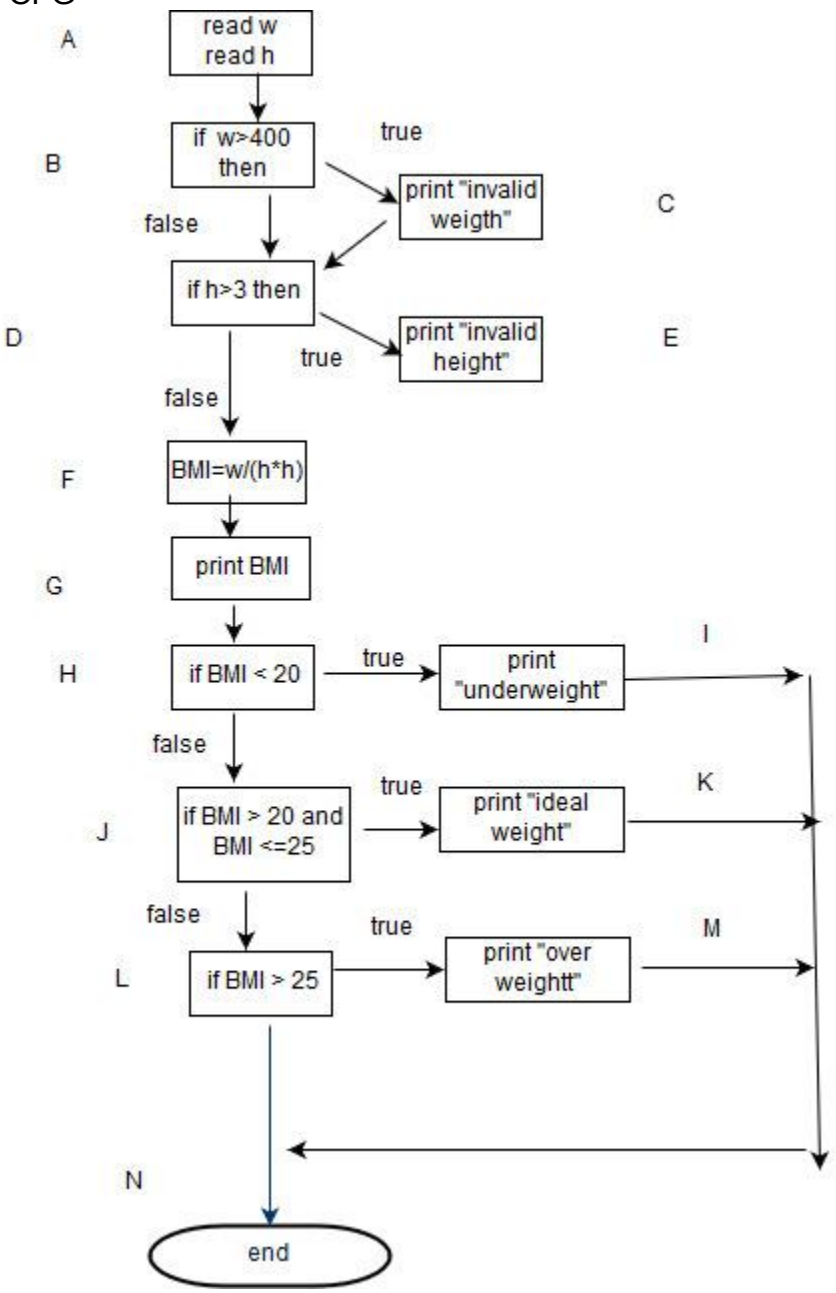
For this pseudocode snippet:

```
1. Read Weight (w)
2. Read Height (h)
3. IF w > 400 THEN
4. Print “invalid weight”
5. ENDIF
6. If h > 3 THEN
7. Print “invalid height”
8. ENDIF
9. BMI=w/(h*h)
10. Print ( “BMI = “ BMI)
11. IF BMI < 20 THEN
12. Print “underweight”
13. ELSE
14. IF BMI >= 20 AND BMI <=25 THEN
15. Print “ideal weight”
16. ELSE
17. IF BMI > 25 THEN
18. Print “overweight”
19. ENDIF
```

a) Draw the control flow graph. [3p]
b) Generate a test suite that achieves 100% statement coverage. [5p]
c) Enhance if necessary your test cases from b) to achieve 100% decision coverage. [3p]
d) Draw a data flow graph. [3p]

e) Generate a test suite that is adequate with respect to the all–uses criterion. [7p]
f) Generate a mutant and show a test case that will kill it. [3p]
g) Generate an equivalent mutant. [3p]

a) CFG

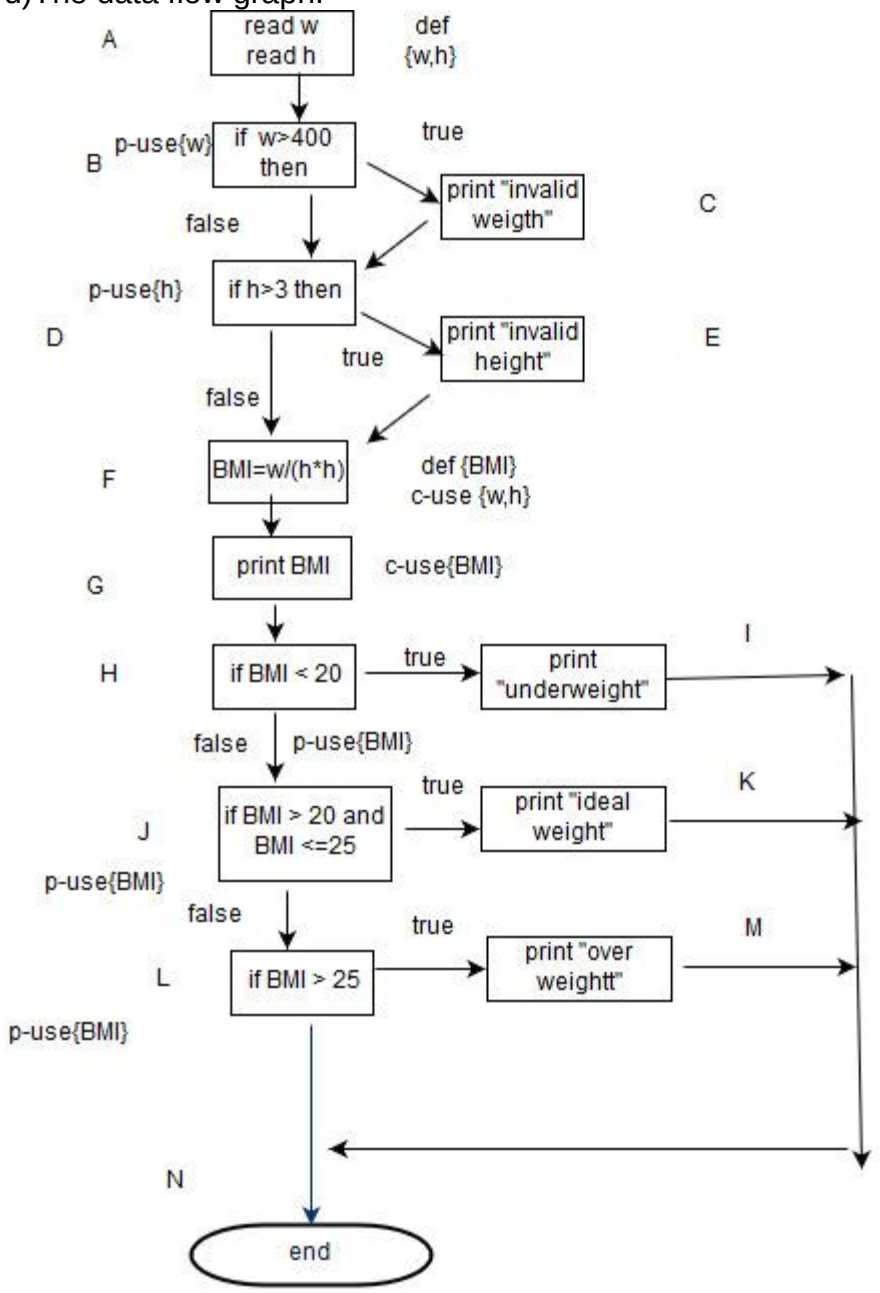


b). these 3 test cases will cover all statements

	w	h	expected output	
TC1	500	5	invalid w, invalid h, 20, ideal weight	
TC2	50	2	12.5 underweight	
TC3	120	1,5	53.3 ideal weight	

c) these 2 TC cover all the decisions

d)The data flow graph.



e)

variable	defined in	c-use	p-use	
w	A	F	{BC,BD}	
h	A	F	{DE, DF}	
BMI	F	G	{HI, HJ}, {JK,JL} {LM,LN}	

The test cases from b) will cover all uses.

	w	h	expected output	path
TC1	500	5	invalid w, invalid h, 20, ideal weight	ABCDEFGHJKN
TC2	50	2	12.5 underweight	ABDFGHIN
TC3	120	1,5	53.3 ideal weight	ABDFGHLMN

f) Many mutants are possible. for example in line 11. If BMI >20.
g) an equivalent mutant cannot be killed because it is semantically the same as the original code. For example in line17. If BMI>=25 instead of if BMI>25.

Appendix. A list with 3 orthogonal arrays.

```
2^3      n=4
000
011
101
110


2^4 4^1      n=8
00000
00112
01011
01103
10013
10101
11002
11110


3^4      n=9
0000
0121
0212
1022
1110
1201
2011
2102
2220
```