ECE 322

SOFTWARE TESTING AND MAINTENANCE

FINAL EXAMINATION
SOLUTIONS

December 17, 2018 9:00 AM

Name (please print)_____
ID _____

Certification: I certify that when working on the final examination, I have not communicated/worked with any student in the ECE 322 class.

Signature:

GOOD LUCK!

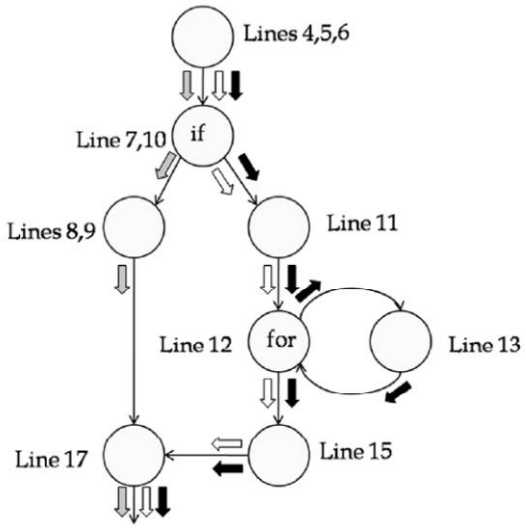
1	2	3	4	5	6	Σ
10	**10**	**10**	**12**	**18**	**10**	**70**

1. [10 points] (i) Given is the following code

```
1 #include <stdio.h>
 2 main()
 3 {
 4   int i, n, f;
 5   printf("n = ");
 6   scanf("%d", &n);
 7   if (n < 0) {
 8     printf("Invalid: %d\n", n);
 9     n = -1;
10   } else {
11     f = 1;
12     for (i = 1; i <= n; i++) {
13       f *= i;
14     }
15     printf("%d! = %d\n", n, f);
16   }
17   return n;
18  }
```

(i) draw a control flow graph
(ii) suggest test cases that realize statement coverage.
(iii) suggest test cases that realize branch coverage.

Solution



statement coverage : test cases n<0 and n> 0
branch coverage: expand the test cases above by bringing the case n==0

2 [10 points] **(i)** For the code shown below do the following:

(a) draw a control flow graph

(b) show all c-use paths.

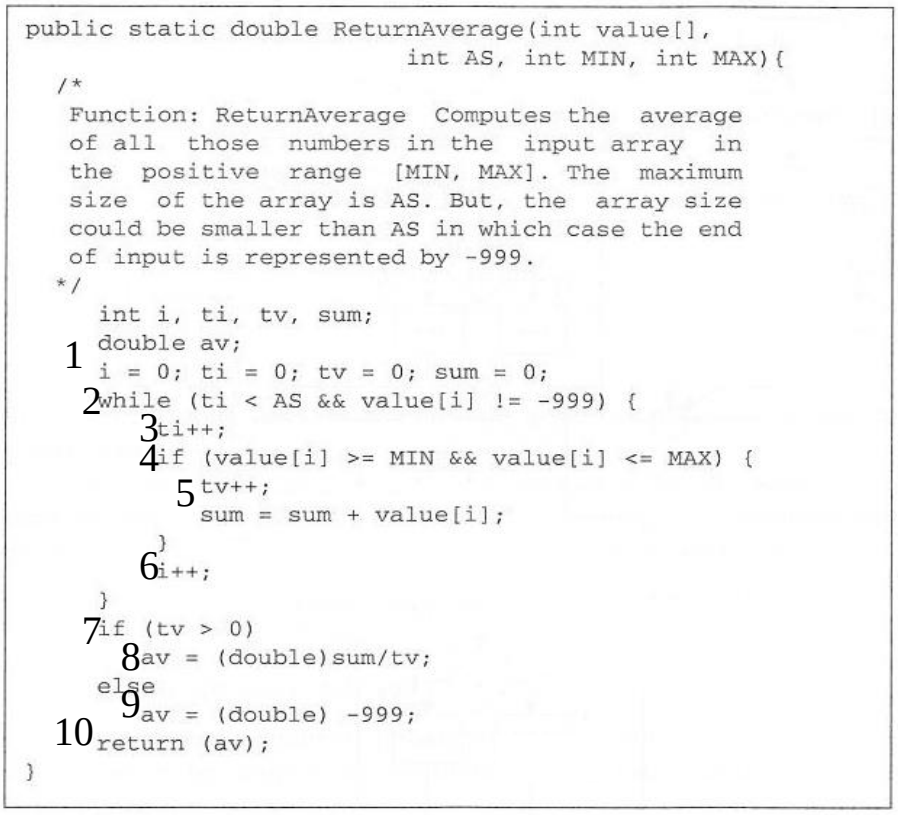
```
public static double ReturnAverage(int value[],
                           int AS, int MIN, int MAX){
  /*
   Function: ReturnAverage  Computes the  average
   of all  those  numbers in the  input array  in
   the  positive  range  [MIN, MAX]. The  maximum
   size  of the array is AS. But, the  array size
   could be smaller than AS in which case the end
   of input is represented by -999.
  */
     int i, ti, tv, sum;
     double av;
     i = 0; ti = 0; tv = 0; sum = 0;
     while (ti < AS && value[i] != -999) {
         ti++;
         if (value[i] >= MIN && value[i] <= MAX) {
            tv++;
            sum = sum + value[i];
         }
         i++;
     }
     if (tv > 0)
        av = (double)sum/tv;
     else
        av = (double) -999;
     return (av);
}
```

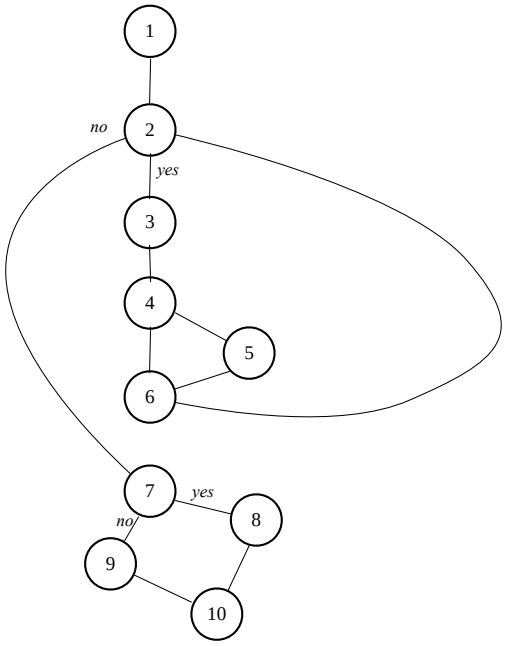
In your solution, use the following line numbering

```
public static double ReturnAverage(int value[],
                        int AS, int MIN, int MAX){
  /*
   Function: ReturnAverage  Computes the  average
   of all  those  numbers in the  input array  in
   the  positive  range  [MIN, MAX]. The  maximum
   size  of the array is AS. But, the  array size
   could be smaller than AS in which case the end
   of input is represented by -999.
  */
     int i, ti, tv, sum;
     double av;
1    i = 0; ti = 0; tv = 0; sum = 0;
2    while (ti < AS && value[i] != -999) {
3       ti++;
4       if (value[i] >= MIN && value[i] <= MAX) {
5          tv++;
           sum = sum + value[i];
        }
6       i++;
     }
7    if (tv > 0)
8       av = (double)sum/tv;
     else
9       av = (double) -999;
10   return (av);
}
```

Solution

Control flow graph

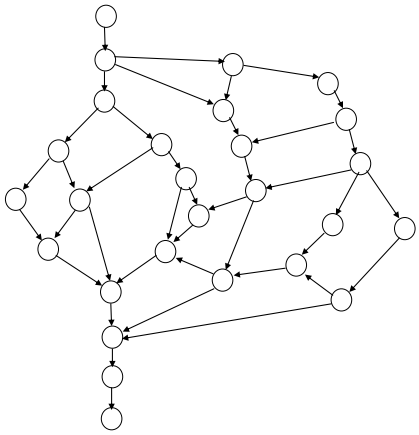




c-use paths (starts with a def and ends in a statement where it is involved in computing)

sum 1-2-3-4-5 5-6-2-7-8
av 8-10 9-10
increment i 1-2-3-4-5-6
increment tv 1-2-3-4-5
ti 1-2-3 3-4-5-6-2-3 3-4-6-2-3

(ii) What is the cyclomatic complexity of the following control flow graph?



Solution
Number of planar regions in the graph = 20

3 [10 points] (i) Using the modified condition/branch coverage criterion, develop a suite of test cases for the following composite condition

a||((!b)&&c)

Solution

Test #	abc	f
1	FFF	F
2	FFT	T
3	FTF	F
4	FTT	F
5	TFF	T
6	TFT	T
7	TTF	T
8	TTT	T

Variable a: 1-5 variable b: 2-4 variable c: 1-2
Test cases: 1, 2, 4, 5

(ii) How many test cases are required when using the weak *n* x 1 testing strategy for the boundary in the 5 dimensional space (x_1, x_2, x_3, x_4, x_5) described as follows

$$|x_1-2| + |x_2-3| + |x_3+1| +|x_4+2| +|x_5+1| \leq 10$$

Could you generalize the result when dealing with an *n*-dimensional space (x_1, x_2, …, x_n)

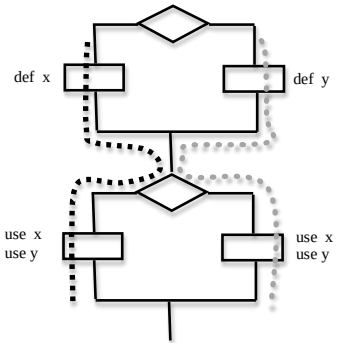
Solution

n =2 : 4 line segments ++ +- -+ -- for each line 3 points (2+1) overall $(2+1)2^2$ test cases
n = 3: 8 planes +++ …--- (2^3) for each plane 4 points (3+1) $(3+1)2^3$ test cases
…
n =5: 2^5 hyperplanes (5+1) points, overall $(5+1)2^5 = 6*32 = 192$
in general $(n+1)2^n$

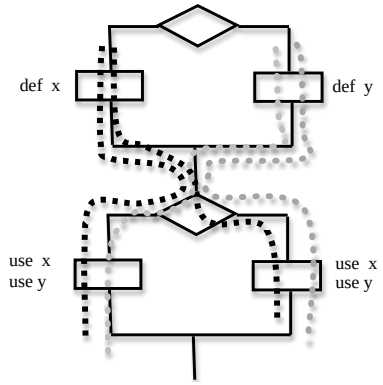
4 [12 points; each question is worth 2 points]
Brief answers: please be concise. A few lines of reply and a sound justification (if required) are just fine.

(1) What is meant by the gap between all-branches testing and all-path testing and how does data flow testing fill the gap?

Solution
Typically, the number of tests resulting from data flow testing is in-between the number of tests generated by the all branching testing and all-path testing. A typical example is shown below -



branch coverage – we require two tests



data flow testing; we need to test all def –use paths (in this case it coincides with tests cases for path coverage)

(2) How does the cyclomatic complexity measure relate to testing coverage criteria?
Solution

Cyclomatic complexity – McCabe complexity measure gives an upper bound on the number of linearly independent paths in the code, viz. the number of tests required.

(3) What software complexity measure describing code could you consider as a sound descriptor of the process of software maintenance?

Solution
Software cyclomatic complexity measure v(G) (as more code is being added and the structure of code changes; commonly deteriorates). We observe here some continuity; no drastic changes are visible.
More critical is essential complexity ev(G) whose values could increase abruptly because of some structural changes encountered in the modified code.

(4) Is it always true that the Extreme Point (EP) strategy leads to more test cases than the weak $n \times 1$ strategy? Justify your answer.

solution

Not always; if there are a large number of linear bounds (b) of the boundary, viz.

$$(n+1)b + 1 > 4^n + 1$$

(5) Identify the fundamental difference between the control flow testing and data flow testing. In which sense are these two categories of testing complementary?

solution
In control flow testing we are concerned with testing how control in the program is realized and what possible faults could be present; we have various coverage criteria (statement, branch, path…). The quality of code could be also quantified; see MCabe complexity measure.
Data flow coverage is focused on identifying possible problems with data; there are several categories of problems present there (defining, re-defining, using…)

(6) The software is to be developed in a very unstable environment. Which integration strategy would you recommend? Why?

solution
top-down; as the environment is unstable, we avoid premature commitment to unstable interfaces

5 [18 points; each question is worth 2 points]
Circle only one answer for each of the questions posed below. If you find any question that appears to have more than one correct answer, choose the one that is the most specific or appropriate.

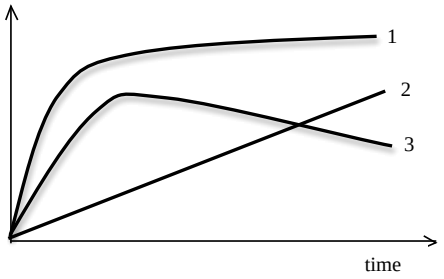
1.Which of the following is a well-known maxim regarding program testing:
(a) only incompetent programmers need to do testing
(b) an exhaustive test is the only practical way of constructing a correctness proof
(c) <u>program testing can be used to discover the presence of errors but not their absence</u>
(d) none of the above is a correct answer

2.The essence of program testing is to choose certain components of the program to be exercised during the test. In data-flow testing, the components to be exercised are:
(a) data structures in the program
(b) paths in the data-flow diagram
(c) <u>segments of an execution path that starts with a definition of some datum and ends with the use of that definition</u>
(d) segments of an execution path that starts with a definition of some datum and ends with the next definition of that datum
(e) none of the above is a correct answer

3.The disadvantages of top-down integration testing include
(i) it may sometimes be difficult to find top-level input data that will exercise a lower-level module in a particular manner desired
(ii) the evolving system may be very expensive to run as a test harness for new routines
(iii) it may be costly to relink and re-execute a system each time a new routine is added

(a) <u>all</u>
(b) all but (i)
(c) all but (ii)
(d) all but (iii)
(e) none of (a)-(d) is a correct answer

4.What are represented by the three curves in the figure below
(a) 1: total cost of testing 2: risk reduction 3: net benefit of testing
(b) 1: risk reduction 2: net benefit of testing 3:total cost of testing
(c) <u>1: risk reduction 2: total cost of testing 3; net benefit of testing</u>
(d) 1: net benefit of testing 2: risk reduction 3: total cost of testing
(e) none of the above (a) – (d)



5.In practice, existing program testing methods are such that

(a) <u>their capabilities complement one another</u>
(b) some have error-detection capabilities identical to others
(c) some can be used to obtain a direct proof of program correctness
(d) they are easy to implement and economical to use
(e) none of the above is a correct answer

6. Suppose that C++ programmer mistakenly writes

```
if (B)
   S1;
   S2;
```

instead of

```
if (B) {
   S1;
   S2;
      }
```

Which of the following statements are true?
(i) it needs only one test to do a statement coverage test for both programs
(ii) it requires at least two test cases to do a branch coverage test for both programs
(iii). The error will never be reflected in the test result if only one test case is used to do the statement coverage test

(a) <u>all</u>
(b) (i) and (ii) only
(c) (i) and (iii) only
(d) (ii) and (iii) only
(e) None of (a) through (d) is a correct answer

7. The presence of data flow anomaly indicates that
(i) the program is definitely in error
(ii) the program is possibly in error
(iii) a datum is defined and then used repeatedly without being redefined

Which of the foregoing statement are true?

a. All
b. All but (i)
c. All but (ii)
d. All but (iii)
e. <u>None of (a) through (d) is a correct answer</u>

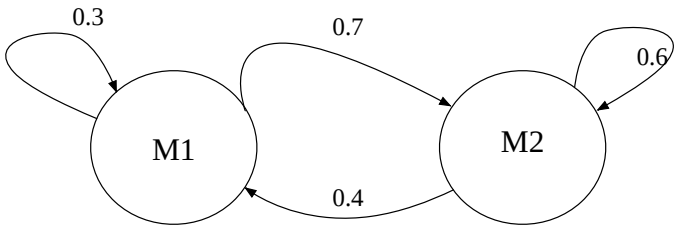
8.A test driver (driver) in integration testing is:

(a) <u>a simulator of a calling program (dummy program unit) used in a bottom-up integration testing</u>
(b) a program unit designed to feed the test cases automatically

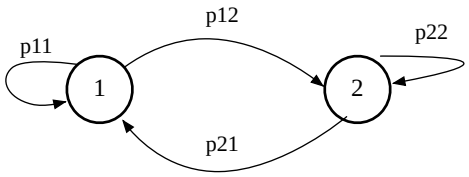
(c) a special input/output routine used to control the input device
(d) a simulator of a calling program (dummy program unit) used in a top-down integration testing
(e) a program designed to generate test data automatically

9.A stub used in integration testing is:
(a) a driver used in a unit test
(b) a subprogram designed to generate and feed the test cases
(c) <u>a simulator of a called program (dummy program unit) used in a top-down integration testing</u>
(d) a program segment designed to prevent a "runaway" program from damaging other programs during a test
(e) a simulator of a calling program (dummy program unit) used in a bottom-up integration test

6 [10 points]
(i)The software system is composed of two modules and the interaction between them is modeled by a Markov model with the transition values shown below. If you were to test the modules, which one would you test first. Justify your choice.



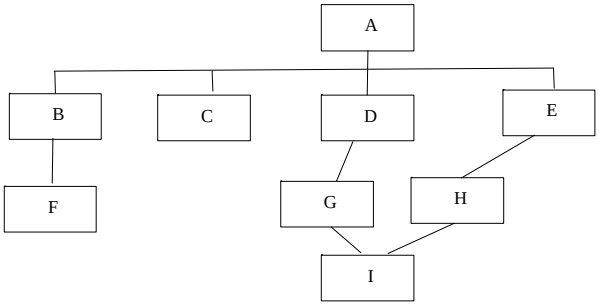
Solution



$p_1 = p_{11}p_1 + p_{21}p_2$
$p_2 = p_{12}p_1 + p_{22}p_2$

Solving this system of equations (note that we need to replace one of them by $p_1+p_2=1$)
We have $p_1 = 0.36$ $p_2 = 0.64$ so module M_2 is to be tested first.

(i) Using the module hierarchy given in the figure below, show the ordering of module integration for the top-down and bottom-up integration approaches. Determine the number of stubs and drivers required for each approach.



Solution

Top down
Test A: stubs B C D E (4)
Test (A B C D E): stubs F G H (3)
Test (A, B, …,D E H): stub I (1)

Total 8 stubs

Bottom up

Test F driver B (1)
Test I: driver G, H (2)
Test I G H driver D E (2)
Test F G H … E : driver A (1)

Total 6 drivers