ASSIGNMENT #3 – SAMPLE SOLUTION

DUE DATE: 7/11/2023 (AT 23.55)

Question #1

Given is the following program

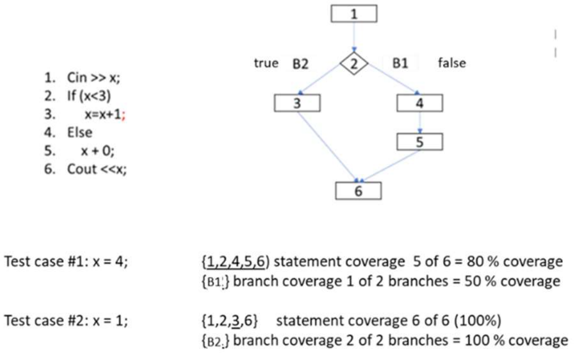
```
void main ()
{
   1.  int n,x,y,i,flag, a;
   2.  cin >> a;   // input a
   3.  cin >> x ;  // input x
   4.  cin >> y;  // input y
   5.  cin >> n  ; // input n
   6.  i=3;
   7.  flag = 1;
                C1           C2                   C3
   8.  while ((i < n) || ((flag <= 0)&& (a!=y)))
       {
   9.     a++;
   10.    y=a;
   //               C4
   11.     if ((a = =y)) then
   12.         {flag= 2;}
   13.    else
   14.       {flag = 2};
   15.     i=i+1;
   16.  }
   //          C6              C7

   17. if ((n > i) && (flag == 0))

   18.   {cout << “We did not find anything”;}  // output

     }
```

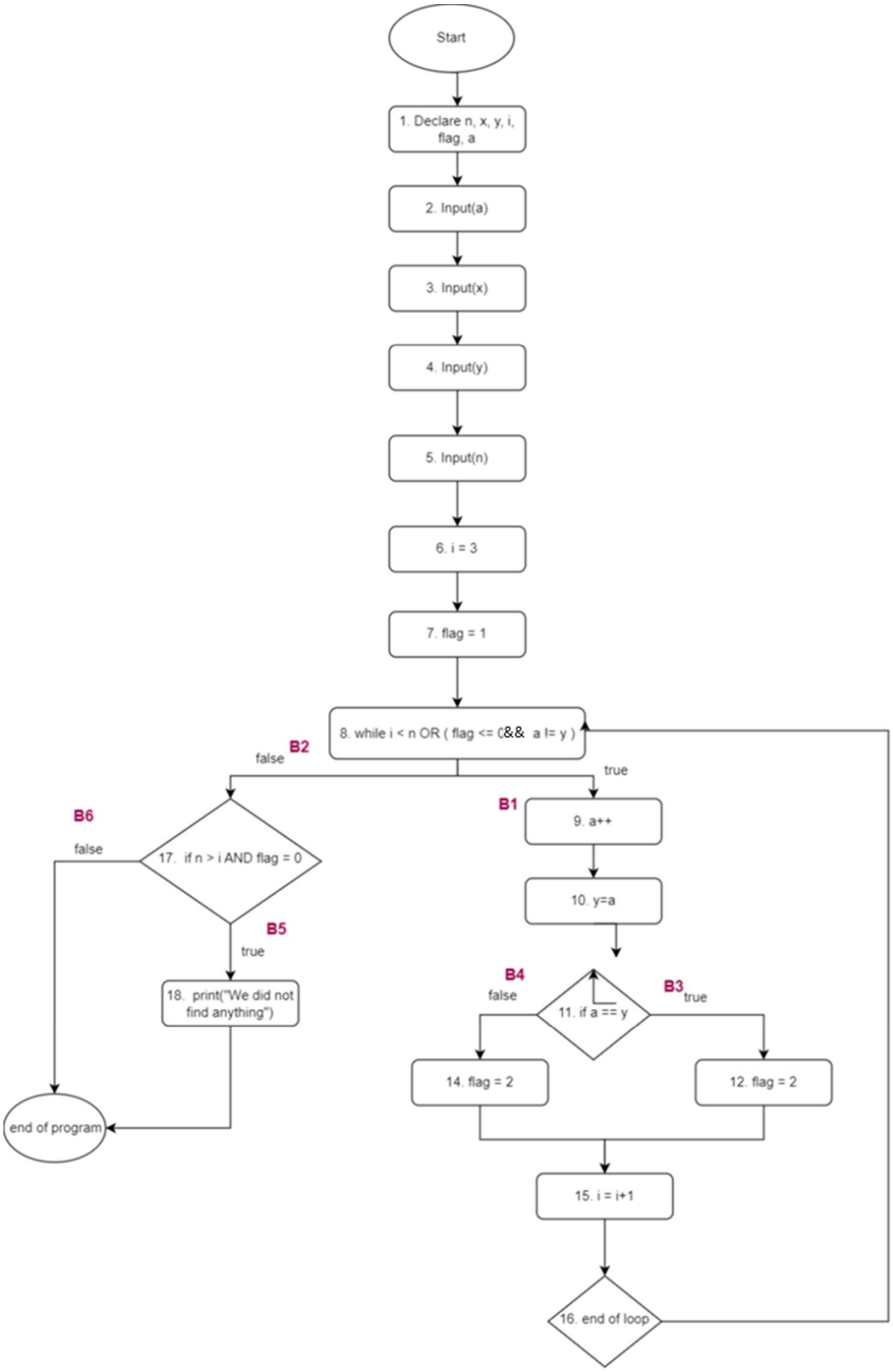
a.) Create a flow diagram and perform branch testing – provide test cases and indicate clearly which statements/branches are covered by your test bases - below an example of a flow diagram. **<u>If a statement/branch/condition cannot be covered by your test case, clearly state which statement/branch/condition and why!</u>**



```
1.  Cin >> x;
2.  If (x<3)
3.      x=x+1;
4.  Else
5.      x + 0;
6.  Cout <<x;
```

Test case #1: x = 4; {1,2,4,5,6) statement coverage 5 of 6 = 80 % coverage
{B1} branch coverage 1 of 2 branches = 50 % coverage

Test case #2: x = 1; {1,2,3,6} statement coverage 6 of 6 (100%)
{B2} branch coverage 2 of 2 branches = 100 % coverage


Start
1. Declare n, x, y, i, flag, a
2. Input(a)
3. Input(x)
4. Input(y)
5. Input(n)
6. i = 3
7. flag = 1
8. while i < n OR (flag <= 0&& a != y)
false
B2
true
B1
9. a++
10. y=a
B6
false
17. if n > i AND flag = 0
B5
true
18. print("We did not find anything")
end of program
B4
false
11. if a == y
B3
true
14. flag = 2
12. flag = 2
15. i = i+1
16. end of loop

Test Case 1:

Path from Start to End (will not enter the loop)

Input a=5, x=10 y =15, n=2

Statements executed: 1,2,3,4,5,6,7,8,17 = Coverage: 9/17 = 53%

Branches covered B2, B6 => 2/6 = 33.33%

Test Case 2:

Execution enters the loop

Input a=5, x=10, y=15, n=5

Statements executed: 1,2,3,4,5,6,7,8,9,10,11,12,15, 8, 17 = Coverage: 14/17 = 82%

Branches covered: B1, B3, B2, B6 = Coverage: 4/6 = 66.66%

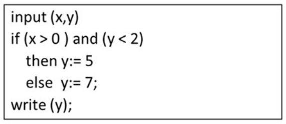
Final testing results:

Statement coverage achieved: 14/17 = 82%

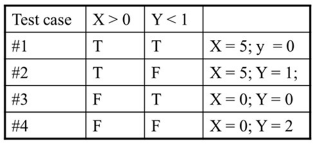
Branch coverage achieved: 4/6 = 66.666%

No test cases can be created to cover the following statements: 13,14,18

No test cases can be created to cover the following branches: B3, B5 b.) Perform multiple condition testing for the program in Question #1. Test cases should be presented as follows.

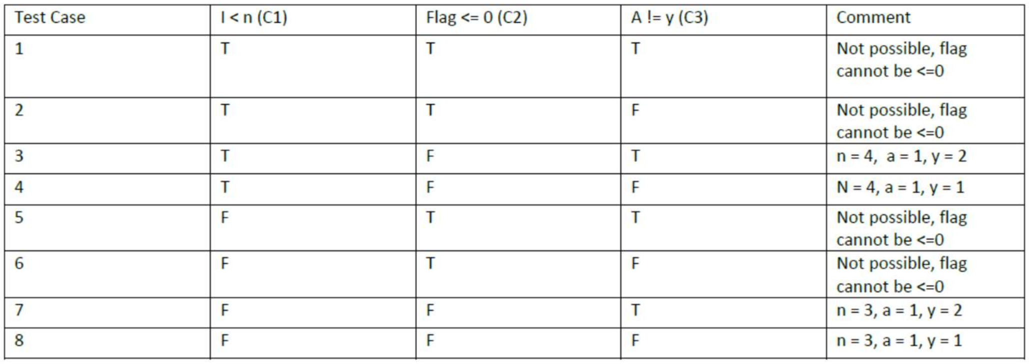


```
input (x,y)
if (x > 0 ) and (y < 2)
   then y:= 5
   else  y:= 7;
write (y);
```



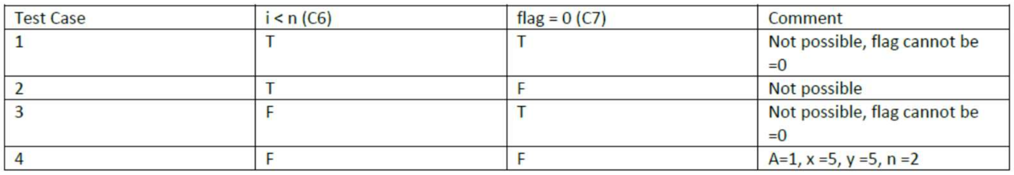
Test case	X > 0	Y < 1	
#1	T	T	X = 5; y = 0
#2	T	F	X = 5; Y = 1;
#3	F	T	X = 0; Y = 0
#4	F	F	X = 0; Y = 2

For statement 8:



Test Case	I < n (C1)	Flag <= 0 (C2)	A != y (C3)	Comment
1	T	T	T	Not possible, flag cannot be <=0
2	T	T	F	Not possible, flag cannot be <=0
3	T	F	T	n = 4, a = 1, y = 2
4	T	F	F	N = 4, a = 1, y = 1
5	F	T	T	Not possible, flag cannot be <=0
6	F	T	F	Not possible, flag cannot be <=0
7	F	F	T	n = 3, a = 1, y = 2
8	F	F	F	n = 3, a = 1, y = 1

For statement# 17:



Test Case	i < n (C6)	flag = 0 (C7)	Comment
1	T	T	Not possible, flag cannot be =0
2	T	F	Not possible
3	F	T	Not possible, flag cannot be =0
4	F	F	A=1, x =5, y =5, n =2

Question #2

Given is the following statement: your new boss states the following: “Given that our organization is using a test-driven development approach, where we write black-box test cases prior to developing our software, there is no more need for white-box testing at the statement coverage criteria, since all the black-box test cases will automatically exercise all statements.

Do you agree/disagree with the statement?

Briefly justify your answer (1-2 sentences).

Short answer: Disagree, because:

Coverage and Depth: Black-box testing may ensure that the software functions as expected from a user's perspective, but it doesn't guarantee that all code paths or edge cases are covered. White-box testing is essential for achieving statement coverage and exploring the intricacies of the codebase.

Code Quality: White-box testing can uncover issues related to code quality, such as code vulnerabilities, unused code, or logical errors that black-box testing might miss. These issues can have a significant impact on the software's reliability and security.

Long answer:

Disagree: The idea that writing black-box test cases alone eliminates the need for white-box testing, particularly at the statement coverage level, is not accurate. Here's why:

Different Focus: Black-box testing focuses on testing the functionality of the software from an external perspective without knowledge of the internal code structure. White-box testing, on the other hand, examines the internal code structure and logic. These two approaches serve different purposes.

Coverage and Depth: Black-box testing may ensure that the software functions as expected from a user's perspective, but it doesn't guarantee that all code paths or edge cases are covered. White-box testing is essential for achieving statement coverage and exploring the intricacies of the codebase.

Code Quality: White-box testing can uncover issues related to code quality, such as code vulnerabilities, unused code, or logical errors that black-box testing might miss. These issues can have a significant impact on the software's reliability and security.

Robustness and Edge Cases: White-box testing is essential for exploring and testing boundary conditions, exceptional cases, and complex decision points in the code. These are often missed by black-box testing.

Verification and Validation: White-box testing helps in verifying the code's correctness and validating its compliance with coding standards and best practices, which can be critical for maintainability and long-term success.

While black-box testing is valuable for validating the software's functionality and user experience, white-box testing is equally important for ensuring the internal quality of the code. A comprehensive testing strategy often includes both black-box and white-box testing, among other testing techniques, to provide a more complete assessment of the software's quality and robustness.