CS 589 - Assignment 3

Sukanta Sharma (A20472623)
CS 589 – Software Testing and Analysis | Fall 2021 | Illinois Institute of Technology
11/22/21

PROBLEM#1 (35 points): Testing polymorphism

For the following function F() and the inheritance relationships between five classes side, A, B, C, and D, design a set of test cases using polymorphic testing, i.e., for each polymorphic call all bindings should be "executed/tested" at least once. For each test case show which binding of the polymorphic call(s) is "executed". Notice that statements, where polymorphic calls are made, are highlighted in bold.

```
1:      int F(int a, int b, int c, int d){
        side *pa, *pb, *pc, *t;

2:          pa=new A;
3:          pc=new C;
4:          pa->set(a);
5:          pc->set(c);
6:          if (pa->get() < pc->get()) {
7:                  t = pa;
8:                  pa = pc;
9:                  pc = t;
            }
10,11:      if (d<0) pb=new D;
12:         else pb=new B;
13:         pb->set(b);
14:         if (pa->get() > pc->get()) {
15:                 t = pa;
16:                 pa = pb;
17:                 pb = t;
            }
18:         if (pa->get() > pb->get()) {
19:                 t = pc;
20:                 pc = pb;
21:                 pb = t;
            }
22:         if (pa->get() + pc->get() <= pb->get())
23:                 return 0;
24:         else return 1;
        }
```

```
class side {
public:
virtual void set(int y) {x=y;};
virtual   void set_x(int y) {x=y;};
virtual   int get(){return x;};
private:
int x;
};

class A: public side {
public:
void set(int y) {if (y<10) set_x(10); else set_x(y);};
};

class B: public side {
public:
void set(int y) {if (y<25) set_x(25);else set_x(y);};
};

class C: public side {
public:
void set(int y) {if (y<0) set_x(0); else set_x(y);};
};

class D: public B {
public:
int get()  {if (side::get()<0) return 0;
            else return side::get();}
};
```

A sample test case: Test #1: a=4, b=7, c=6, d=1

Ans:

There are 9 places where dynamic object binding exists. The following table shows the line polymorphic calls and possible bindings of those.

Line no.	Statement	Binding	Test Case Covered
6	pa → get()	Object of A	Test#1
	pc → get()	Object of C	Test#1
14	pa → get()	Object of A	Test#1
		Object of C	Test#2
	pc → get()	Object of A	Test#2
		Object of C	Test#1
18	pa → get()	Object of A	Test#4
		Object of B	Test#1
		Object of C	Impossible
		Object of D	Test#3
	pb → get()	Object of A	Test#1
		Object of B	Test#4
		Object of C	Test#2
		Object of D	Test#5
22	pa → get()	Object of A	Test#4
		Object of B	Test#1
		Object of C	Impossible
		Object of D	Test#3
	pb → get()	Object of A	Test#2
		Object of B	Test#4
		Object of C	Test#1
		Object of D	Test#5
	pc → get()	Object of A	Test#1
		Object of B	Test#6
		Object of C	Test#2
		Object of D	Test#7

The test cases are given below:

Test Case no.	a	b	c	d
Test#1	0	0	0	0
Test#2	0	0	13	14
Test#3	0	0	13	-14
Test#4	0	0	10	14
Test#5	0	0	10	-14
Test#6	26	0	26	14
Test#7	26	0	26	-14

PROBLEM#2 (35 points): Symbolic evaluation

For the following function *F(int a, int b, int c)* use symbolic evaluation to show that the multiple-condition (True, False) in line 15 is **not executable**, i.e.,

$$\left((a == c) \mid\mid (b == c)\right)$$

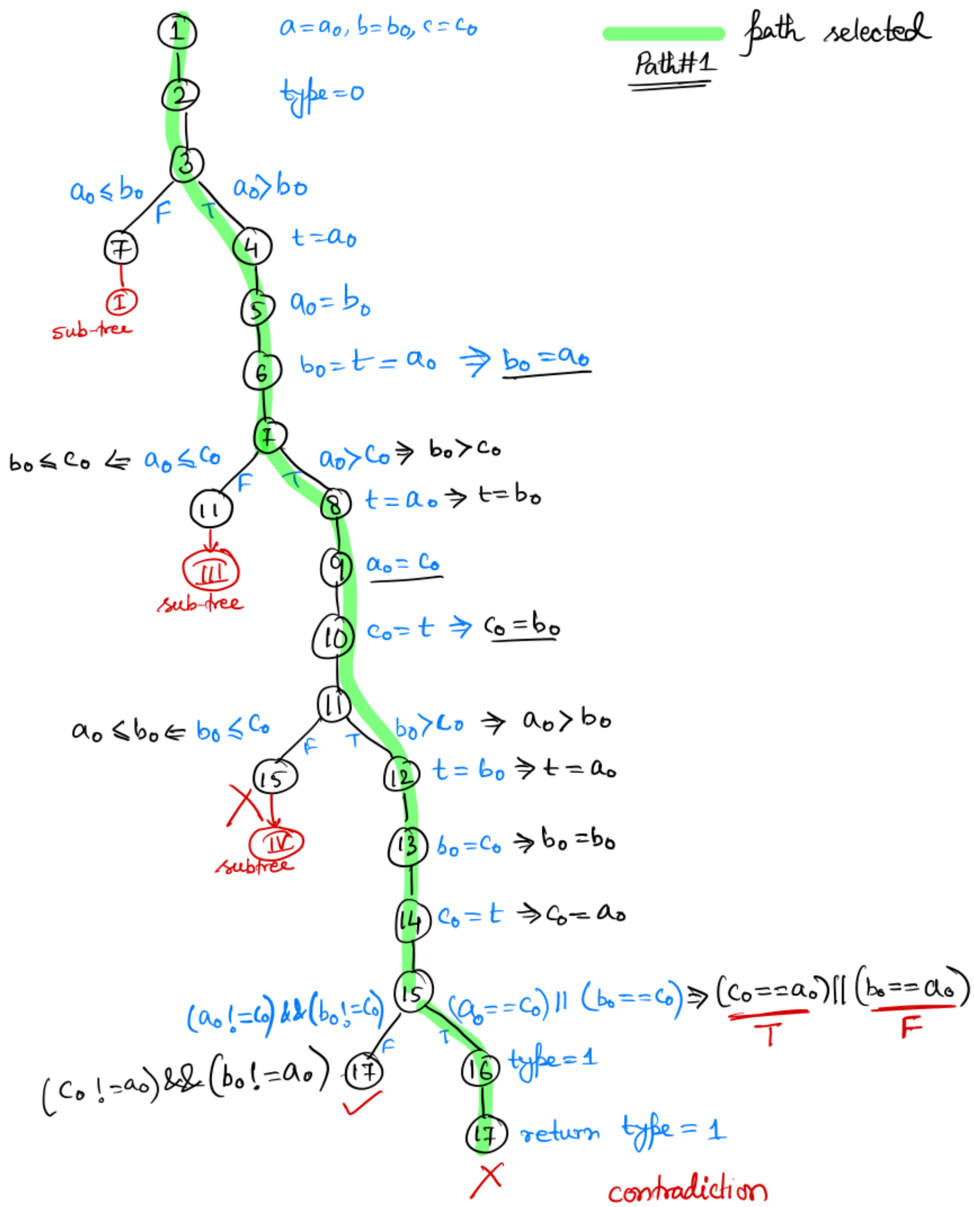
True False

In your solution provide the **symbolic execution tree**.

```
1:    int F(int a, int b, int c)
      { int type, t;
2:          type = 0;
3:          if (a > b) {
4:                t = a;
5:                a = b;
6:                b = t;
            }
7:          if (a > c) {
8:                t = a;
9:                a = c;
10:               c = t;
            }
11:         if (b > c) {
12:               t = b;
13:               b = c;
14:               c = t;
            }
15:         if ((a == c) || (b == c)) {
16:               type = 1;
            }
17:         return type;
      }
```

Ans:

The symbolic evaluation tree is given below:



This path is not possible for

$c_0 == a_0 \Rightarrow$ True and $b_0 == a_0 \Rightarrow$ False

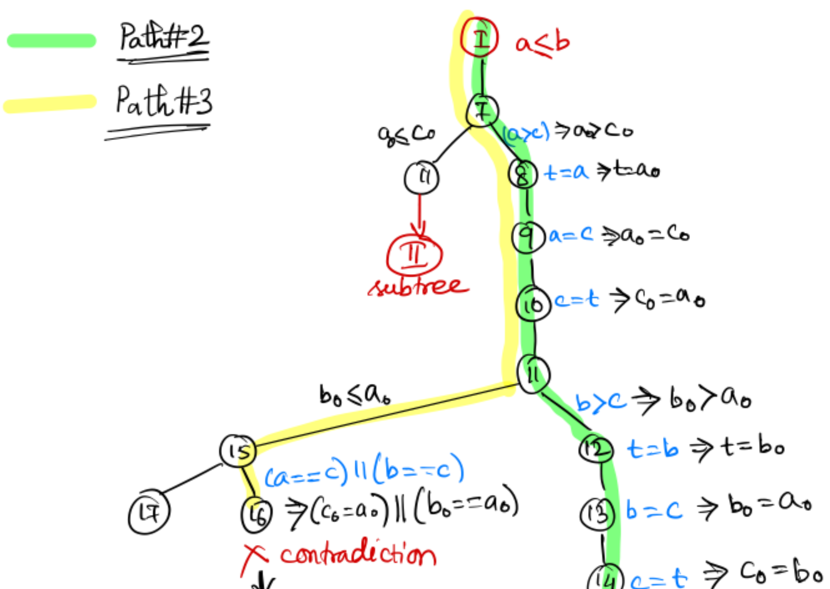
because from path 3→4, we get $a_0 > b_0$

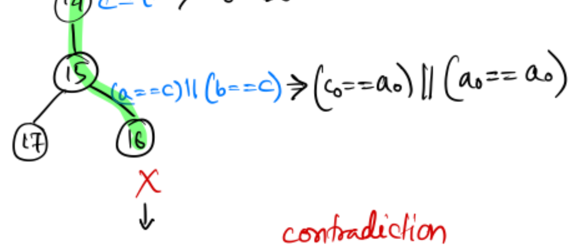
" " 7→8, " " $b_0 > c_0$

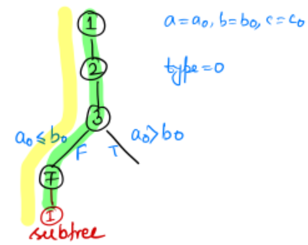
so, $a_0 > b_0 > c_0$

$\therefore$ $c_0 == a \Rightarrow$ False

$b_0 == a_0 \Rightarrow$ False







This path#3 is not possible for

$c_0 == a_0 \Rightarrow$ True and $b_0 == a_0 \Rightarrow$ False

from path $3 \to 7$, we have, $a_0 \leq b_0$

from " $7 \to 8$, " " $a_0 > c_0$

so, $b_0 \geq a_0 > c_0$

from path $11 \to 15$, we have $b_0 \leq a_0$

this is only possible when

$a_0 == b_0 \Rightarrow$ True

Thus, $c_0 == a_0 \Rightarrow$ False

$b_0 == a_0 \Rightarrow$ True

This path#2 is not possible for

$c_0 == a_0 \Rightarrow$ True and $a_0 == a_0 \Rightarrow$ False

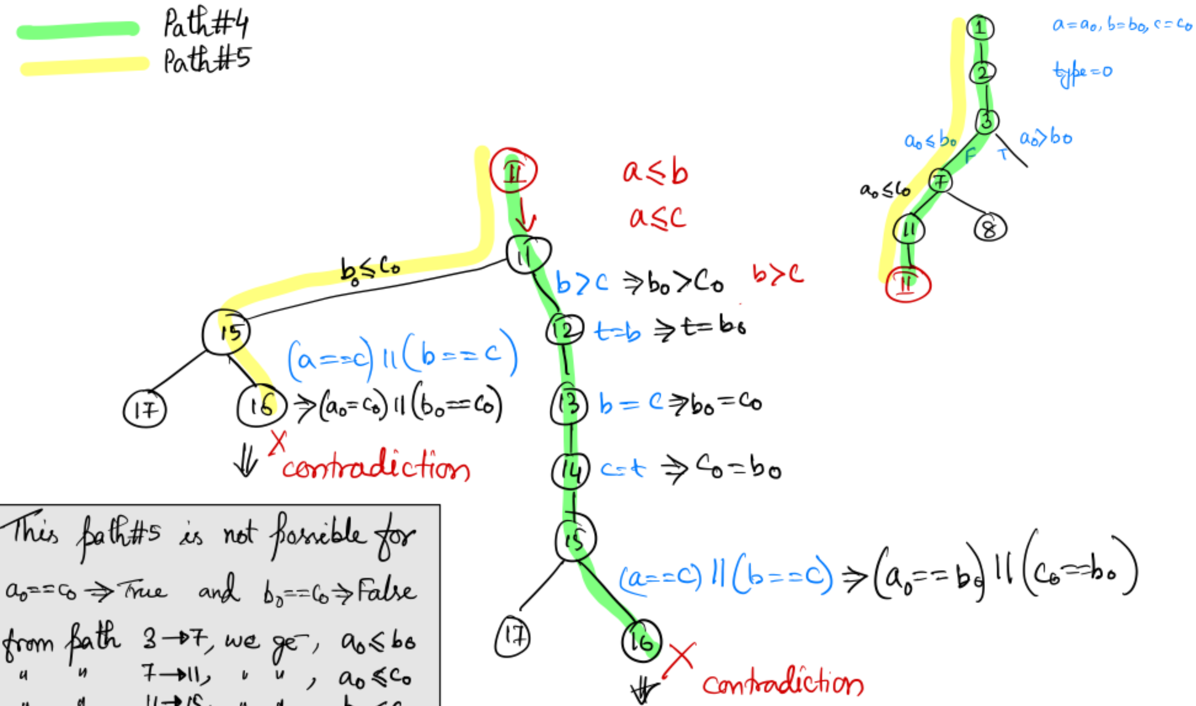
from path $3 \to 7$, we have, $a_0 \leq b_0$

from " $7 \to 8$, " " $a_0 > c_0$

so, $b_0 \geq a_0 > c_0$

Thus, $c_0 == a_0 \Rightarrow$ False

$a_0 == a_0 \Rightarrow$ True

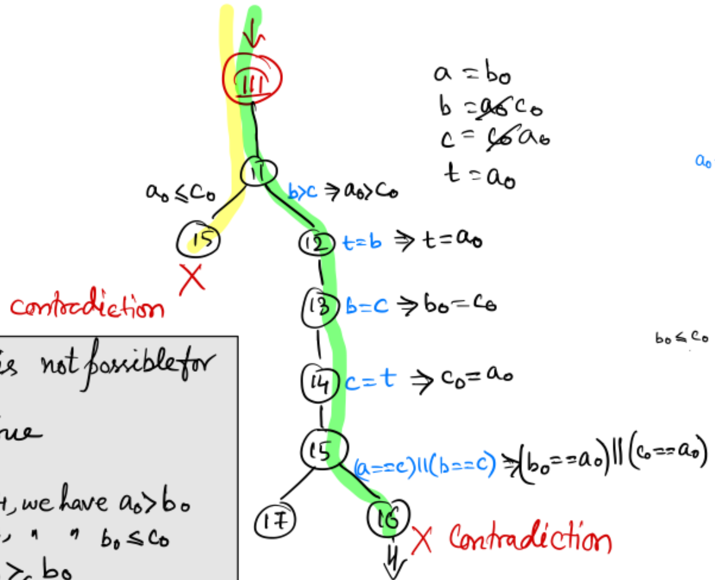


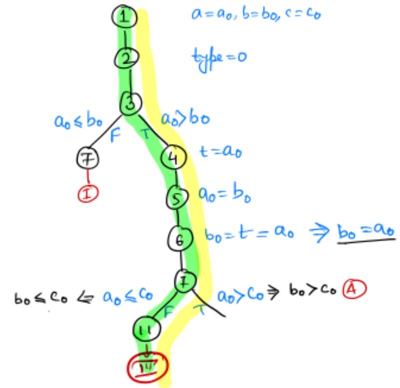
This path#5 is not possible for
$a_0==c_0 \Rightarrow$ True and $b_0==c_0 \Rightarrow$ False
from path $3 \to 7$, we get, $a_0 \leq b_0$
" " $7 \to 11$, " " , $a_0 \leq c_0$
" " $11 \to 15$, " " , $b_0 \leq c_0$
Thus, $a_0==c_0 \Rightarrow$ True
$b_0==c_0 \Rightarrow$ True

This path#4 is not possible for
$a_0==b_0 \Rightarrow$ True and $c_0==b_0 \Rightarrow$ False
from path $3 \to 7$, we get, $a_0 \leq b_0$
" " $7 \to 11$, " " , $a_0 \leq c_0$
" " $11 \to 12$, " " , $b_0 > c_0$
Now, we conclude that $a_0==b_0==c_0$
so, $a_0==b_0 \Rightarrow$ True
$c_0==b_0 \Rightarrow$ True

Path#6

Path#7





This path#7 is not possible for

$a_0 \leq c_0 \Rightarrow$ True

from path 3→4, we have $a_0 > b_0$
" " 7→11, " " $b_0 \leq c_0$
so, $a_0 > c_0 \geq b_0$

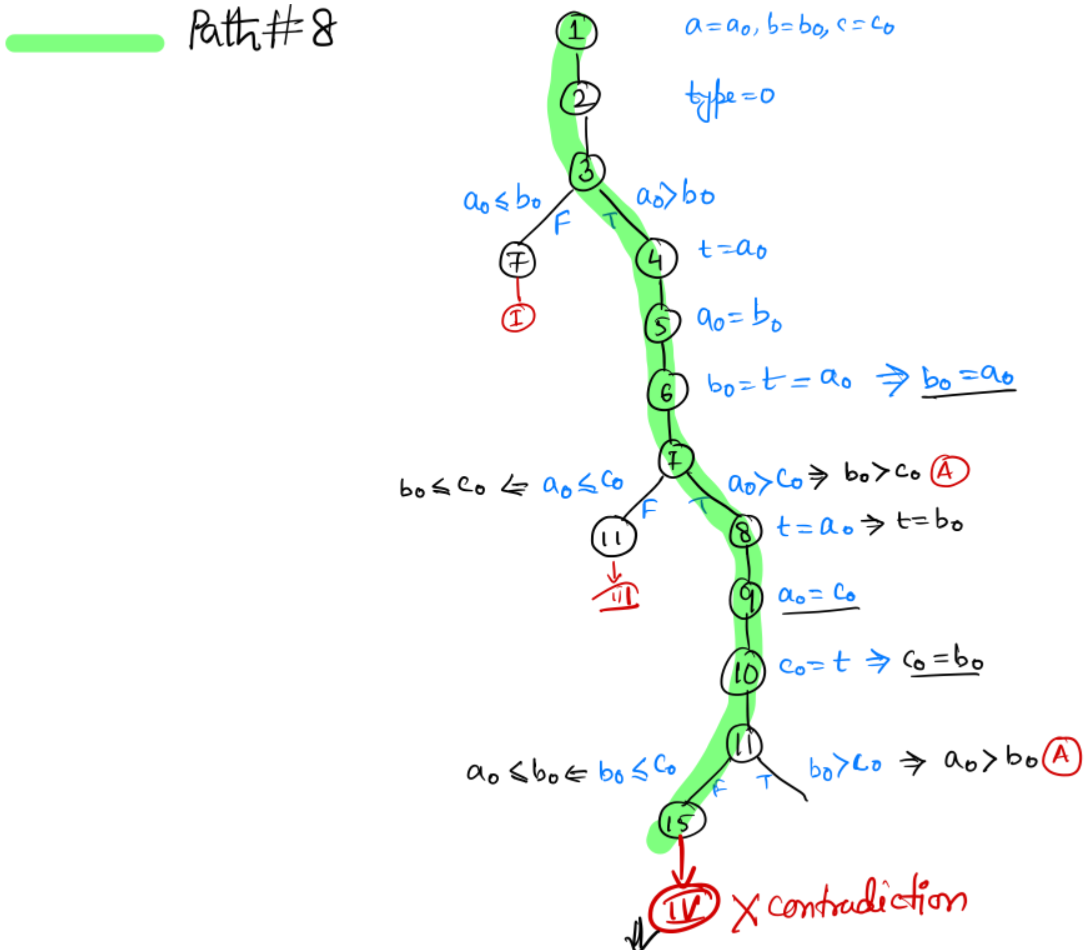
Thus, $a \leq c_0 \Rightarrow$ False

This path#6 is not possible for

$b_0 == a_0 \Rightarrow$ True and $c_0 == a_0 \Rightarrow$ False

from path 3→4, we have $a_0 > b_0$
" " 7→11, " " $b_0 \leq c_0$
so, $a_0 > c_0 \geq b_0$

Thus, $b_0 == a_0 \Rightarrow$ False
$c_0 == a_0 \Rightarrow$ False



This path is not possible for

$a_0 \leq b_0 \Rightarrow$ True

from path $3 \to 4$, we get $a_0 > b_0$

from " $7 \to 8$, " " $b_0 > c_0$

so, $a_0 > b_0 > c_0$

Thus, $a_0 \leq b_0 \Rightarrow$ False

Hence it is proved that line 15 is not executable for

$(a==c) \,||\, (b==c)$

True False

PROBLEM#3 (30 points): Program proving

The following function F() computes the summation of absolute values of elements of the array a[] consisting of n elements. Prove that function F() is correct for the given pre-condition and post-condition:

Pre-condition: $\mathbf{1 \leq n \leq 100}$

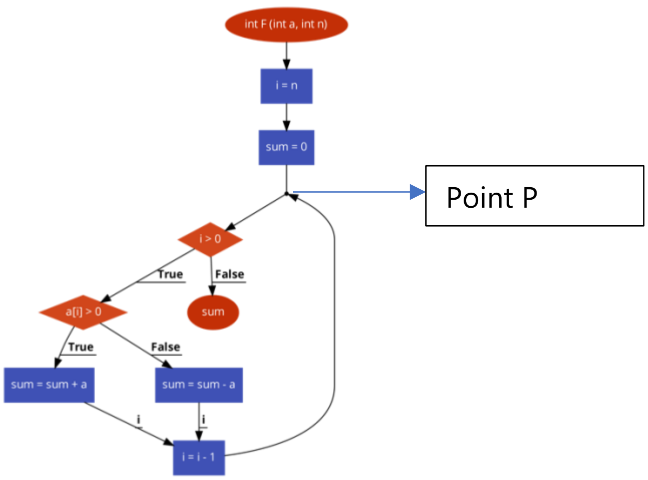
Post-condition:

$$\boldsymbol{sum = \sum_{j=1}^{n} |a[j]|}$$

```
int F (int a[], int n) {
int i, sum;
        i = n;
        sum=0;
        while (i > 0) {
                if (a[i]>0) sum = sum + a[i];
                else sum = sum - a[i];
                i = i - 1;
        };
        return sum;
}
```

Ans:

The flowchart for the given code block is given below:



The line of code from 5 to 8 is basically doing the following equation:

$$sum = sum + |a[i]|$$

This is because if $a[i] > 0$, it has been added to the previous value of sum, and otherwise the value of $a[i]$ is being subtracted from the previous value of sum.

Let's assume $a[i] = x \; if \; a[i] > 0 \; and \; a[i] = -x$ otherwise

$$sum = sum + x, if \; a[i] > 0$$

OR

$$sum = sum - (-x) = sum + x, otherwise$$

To proceed with the proof first we need to find the loop invariant.

K → number of times execution reaches point P

K	i	sum
1	n	0
2	$n-1$	$\lvert a[n] \rvert$
3	$n-2$	$\lvert a[n-1] \rvert + \lvert a[n] \rvert = \sum_{j=n-1}^{n} \lvert a[j] \rvert$
...	...	...
K	$n - K + 1$	$\sum_{j=i+1}^{n} \lvert a[j] \rvert$

$\boldsymbol{K}$	$\boldsymbol{i}$	$\boldsymbol{sum}$
$\boldsymbol{K+1}$	$n-K$	$\sum_{j=n-k+1}^{n} \lvert a[j] \rvert$

So, loop invariant is selected as

$$sum = \sum_{j=i+1}^{n} |a[j]|$$

By mathematical induction we can proof the given statement.

Case 1: Loop Entry (K =1):

$$i = n$$

$$sum = 0 = \sum_{j=n+1}^{n} |a[j]|$$

The sum will be zero, because lower bound for the summation is beyond the range of n, which is invalid, so sum will hold its initial value, i.e., 0. So, the statement is **true** for entry point of the loop.

Case 2: Inside loop for any K :

Assumption:

$$sum_K = \sum_{j=i_K+1}^{n} |a[j]|$$

This is **true**, where:

sum_K, i_K→value of sum and i when execution reaches point P for K^{th} time

We must show and proof the following is also true.

$$sum_{K+1} = \sum_{j=i_{K+1}+1}^{n} |a[j]|$$

Where sum_{K+1}, i_{K+1}→value of sum and i when execution reaches point P for $(K+1)^{th}$ time.

Now,

$$sum_K = \sum_{j=i_K+1}^{n} |a[j]|$$

$$i_{K+1} = i_K - 1$$

$$sum_{K+1} = sum_K + |a[i_K]|$$

$$sum_{K+1} = \sum_{j=i_K+1}^{n} |a[j]| + |a[i_K]| = \sum_{j=i_K+1-1}^{n} |a[j]| = \sum_{j=i_{K+1}+1}^{n} |a[j]|$$

Therefore, for the given assumption the above is **true** for $K + 1$ also.

Case 3: On Termination:

$$i = 0$$

$$sum = \sum_{j=i+1}^{n} |a[j]| = \sum_{j=0+1}^{n} |a[j]| = \sum_{j=1}^{n} |a[j]|$$

The above condition is **true** for the given post-condition

This holds true for any value between $1 \leq n \leq 100$. This proves the correctness of the given function *F()* is correct for the given pre-condition and post-condition.