Assignment 7 (20 points), SE 421, 10/26/2020, due: Monday, 11/2/2020

Name (Last, First): Morellas, Stamatios

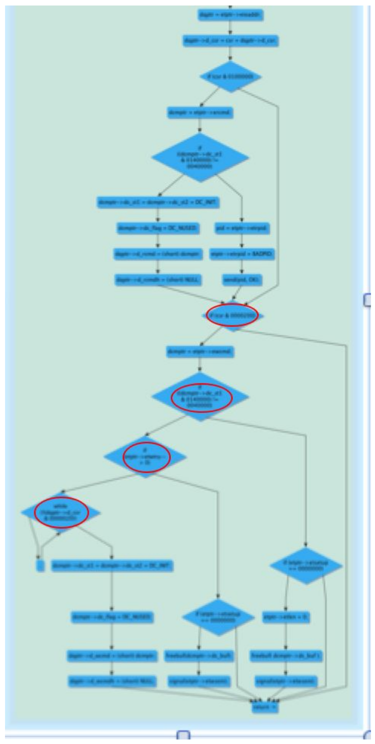
Problem 1 (5 points): For CFG graphs shown below, answer the following:

1. How many governing branch (GB) nodes for the loop in the CFG? Mark each of those nodes as GB.
2. How many differential branch (DB) nodes for the loop in the CFG? Mark each of those nodes as DB.
3. What is the fraction f of the number of paths going through the given loop? (f = n/t where n is the number paths on which the loop can be entered, t is the total number paths)

Note: The governing branches are annotated with red ovals, while the differential branches are annotated with red boxes.

For graph 1:

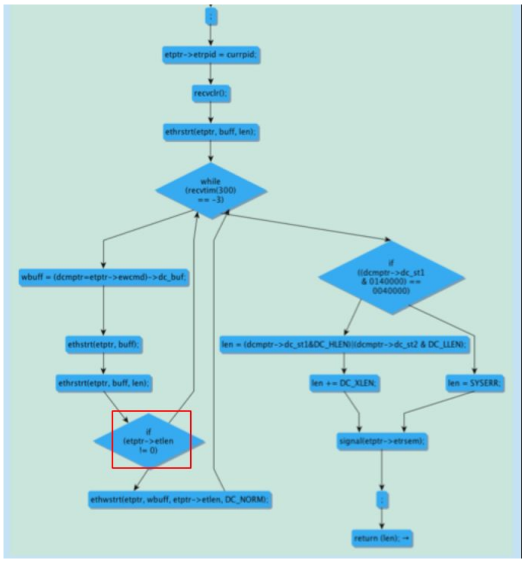
1. There are <u>four</u> governing branch nodes for the loop in the CFG.
2. There are <u>no</u> differential branch nodes for the loop in the CFG.
3. $f = n/t = 3/18$



For graph 2:

1. There are <u>no</u> governing branch nodes for the loop in the CFG.
2. There is <u>one</u> differential branch node for the loop in the CFG.

3. $f = n/t = 1/2$


etptr->etrpid = currpid;
recvclr();
ethrstrt(etptr, buff, len);
while (recvtim(300) == -3)
wbuff = (dcmptr=etptr->ewcmd)->dc_buf;
ethstrt(etptr, buff);
ethrstrt(etptr, buff, len);
if (etptr->etlen != 0)
ethwstrt(etptr, wbuff, etptr->etlen, DC_NORM);
if ((dcmptr->dc_st1 & 0140000) == 0040000)
len = (dcmptr->dc_st1&DC_HLEN)|(dcmptr->dc_st2 & DC_LLEN);
len += DC_XLEN;
len = SYSERR;
signal(etptr->etrsem);
return (len);

Problem 2 (15 points): You are given the Java code for a game app. The app has AC time vulnerability. You play a role in controlling the AC vulnerability. Triggering the vulnerability in a game round causes a program timer to go over limit and award one point to you. One run of the app amounts to 100 game rounds.

Description of the game: Each round of the game results in an ordered list of length 6 of 6 given geometric shapes. In Part I you have control in choosing one shape (and its order) in the list; in Part II you have control over two shapes (and their order) in the list. The remaining shapes in the list are generated randomly.

Exercising control: You exercise control by modifying the given program. With a simple change in the program you can choose the shape and its order in the list. The program as given, has the circle shape to go in the fourth position so that the list is: S 1 , S 2 , S 3 , C, S 5 , S 6 where S i is randomly generated and C is the circle. Note that a shape could repeat.

Part I: Replace the circle by a shape and position of your choice
Part II: Select two shapes in positions of your choice (the two shapes could be same)

Requirements: Grading will be based on how well you fulfill the following requirements:

1. (2 Points) Understand the given program. As a warm-up exercise, without any modification, run the program 10 times. Report the reward points for each run. Note that the reward points should be between 0 to 100. There can be variability in reward points because of the random choices in the program.
    a. **5, 7, 5, 12, 12, 11, 9, 8, 12, 11**
2. (3 Points) Make appropriate modifications. Include in your report, the listing of the modified part of the program for Part I and II.

a. Part I

```
public static void baseExperiement() {
        int rewardsCount = 0; // store the number of rewards gained.

        for(int catalogIndex = 0; catalogIndex < SHAPES_CATALOG_COUNT; catalogIndex++) {

                Shape [] shapes = new Shape[SHAPES_CATALOG_LENGTH];
                for(int i = 0; i < SHAPES_CATALOG_LENGTH; i++) {
                        if(i == 5) {
                                //##############################################
                                // You are only allowed to control the if condition and the shapes to add within this block
                                Shape rectangleShape = new Rectangle();
                                shapes[i] = rectangleShape;
                                //##############################################
                        } else {
                                Shape shape = ShapeUtils.createRandomShape(); // creates a random shape.
                                shapes[i] = shape;
                        }
                }
                ShapesCatalog catalog = new ShapesCatalog(shapes); // constructs a new shapes catalog.
                try {
                        catalog.getStats();
                        //System.out.println(catalog.getStats()); print the stats
                } catch (TimeLimitExceededException e) {
                        rewardsCount++; // We have gained a new reward; increment the rewardsCount.
                }
                catalog.destroy(); // destroy the shapes catalog.
        }
        System.out.println("(Base Experiment) You have gained [" + rewardsCount + "] rewards out of [" +
SHAPES_CATALOG_COUNT + "] rewards!");
    }
```

b. Part II

```
public static void baseExperiement() {
        int rewardsCount = 0; // store the number of rewards gained.

        for(int catalogIndex = 0; catalogIndex < SHAPES_CATALOG_COUNT; catalogIndex++) {

                Shape [] shapes = new Shape[SHAPES_CATALOG_LENGTH];
                for(int i = 0; i < SHAPES_CATALOG_LENGTH; i++) {
                        if(i == 5) {
                                //##############################################
                                // You are only allowed to control the if condition and the shapes to add within this block
                                Shape rectangleShape = new Rectangle();
                                Shape anotherRectangle = new Rectangle();
                                shapes[i] = rectangleShape;
                                shapes[i - 1] = anotherRectangle;
                                //##############################################
                        } else {
                                Shape shape = ShapeUtils.createRandomShape(); // creates a random shape.
                                shapes[i] = shape;
                        }
                }
                ShapesCatalog catalog = new ShapesCatalog(shapes); // constructs a new shapes catalog.
                try {
                        catalog.getStats();
                        //System.out.println(catalog.getStats()); print the stats
                } catch (TimeLimitExceededException e) {
                        rewardsCount++; // We have gained a new reward; increment the rewardsCount.
                }
                catalog.destroy(); // destroy the shapes catalog.
        }
        System.out.println("(Base Experiment) You have gained [" + rewardsCount + "] rewards out of [" +
SHAPES_CATALOG_COUNT + "] rewards!");
    }
```

3. (5 Points) Experiments: Perform experiments to come up with your favorite choice that maximizes your reward points. Once you have your favorite choice, do not change it. Run the program 10 times with that choice. Note that each time the program runs, it is 100 game rounds and thus it can result in up to 100 reward points. Report the award points for each of those 10 runs in the following format:

   Part I:

   The favorite choice: **Rectangle**
   Stats for 10 program runs with the favorite choice:

Program Run	1	2	3	4	5	6	7	8	9	10
Reward Points for the Run	58	56	55	64	63	62	56	66	62	59

   Part II:

   The favorite choice: **Rectangle + Rectangle**
   Stats for 10 program runs with the favorite choice:

Program Run	1	2	3	4	5	6	7	8	9	10
Reward Points for the Run	100	100	100	100	100	100	100	100	100	100

Note: There can be variability in the number of reward points for each program run. This is due to the random choices in the list. With this Table, you can show the variability. You must report the favorite choice and the Table for Parts I and II separately.

4. (5 Points) What is the minimum number of choices that ensures 100 reward points? Explain why. The explanation must be clear and concise. It must clearly identify the root cause of the vulnerability and how it works.

I found that if there are two rectangles in the shapes array at indices 4 and 5, we are rewarded with a count of 100 in the reward points. We know that there is an algorithmic space complexity vulnerability, since there is not a slow execution or crash of the program. Due to the DB node (if statement) inside the second loop, this will execute the same way each time the program is run, which means there is a consistent execution pattern respective to this node. More specifically, the vulnerability seems to occur when setNext gets called for the 4th and 5th indices in the shapes array, causing each rectangle to point to a next rectangle, even if there isn't space in the shapes array for it.

5. Extra Bonus (3 points):
   a. Provide the expected number of reward points for Part I. Give the mathematical analysis that supports the answer.
   b. Provide the expected number of reward points if all six choices are random. Give the mathematical analysis that supports the answer.

NOTE: Making all choices random amounts to the so-called fuzzing. The part #4 is to exemplify the importance of mathematical modeling.

Running the Program: Import the given project zip file in Eclipse and run the program. If you have any doubts, watch the demonstration shown during the lecture.