COSC 3P95- Software Analysis & Testing

Assignment 1

Due date: Monday, Oct 16th, 2023, at **23:59** (11:59 pm)

Delivery method: This is an individual assignment. Each student should submit one PDF through Brightspace.

Attention: This assignment is worth 10% of the course grade. Please also check the Late Assignment Policy.

Name: _**Steve Mastrokalos**____ **Student ID:** **7276900**_____

I highly recommend using my github submission, it is much cleaner looking.

https://github.com/vestex727/COSC-3P95

Questions:

1- Explain the difference between "sound" and "complete" analysis in software analysis. Then, define what true positive, true negative, false positive, and false negative mean. How would these terms change if the goal of the analysis changes, particularly when "positive" means finding a bug, and then when "positive" means not finding a bug. **(10 pts)**

A sound analysis will report all problems/errors/bugs in the code, whereas a complete analysis has no false positives when reporting the problems/bugs/errors.

A true positive is where the analysis (tool) detects a problem/bug/error in the code when there is at least one.

A false positive is where the analysis (tool) finds a problem/bug/error where there is no problem/bug/error

.

A true negative is where the analysis (tool) correctly identifies no problem(s)/bug(s)/error(s).

A false negative is where the analysis (tool) identifies no problem/error/bug when there are problem(s)/bug(s)/error(s).

If we changed the meaning of positive to be not finding a bug, then the following would be the definitions:
A true negative is where the analysis (tool) detects a problem/bug/error in the code when there is at least one.

A false negative is where the analysis (tool) finds a problem/bug/error where there is no problem/bug/error
.
A true positive is where the analysis (tool) correctly identifies no problem(s)/bug(s)/error(s).

A false positive is where the analysis (tool) identifies no problem/error/bug when there are problem(s)/bug(s)/error(s).

2- Using your preferred programming language, implement a random test case generator for a sorting algorithm program that sorts integers in ascending order. The test case generator should be designed to produce arrays of integers with random lengths, and values for each sorting method.
A) Your submission should consist of:
   a. Source code files for the sorting algorithm and the random test case generator.
   b. Explanation of how your method/approach works and a discussion of the results (for example, if and how the method was able to generate or find any bugs, etc.). You can also include bugs in your code and show your method is able to find the input values causing that.
   c. Comments within the code for better understanding of the code.
   d. Instructions for compiling and running your code.
   e. Logs generated by the print statements, capturing both input array, output arrays for each run of the program.
   f. Logs for the random test executions, showing if the test was a pass and the output of the execution (e.g., exception, bug message, etc.).

3-
```java
/**A silly, semi functional sorting algorithm made by yours truly
 *
 *@version 1.0 (Oct 15th, 2023)
 *@author Steve Mastrokalos   */

import java.util.Arrays;

public class SortingAlgorithm {

    /** A sorting algorithm that takes in an array of integers of any
size and returns a sorted array
     *
     * @param array input array to be sorted
     * @return  sorted array
     */
    public static int[] SortArray(int[] array){
        Arrays.sort(array, 1, array.length-1);
        return array;
    }
}


/**A program that tests sorting algorithms and puts the created array
```

```
into a txt file, and the sorted array into another
 * Tests if array is properly sorted
 * Details in console and in sorted text file
 *
 *@version 1.0 (Oct 15th, 2023)
 *@author Steve Mastrokalos   */

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class TestHarness {

    FileWriter writer;
    File output;
    static final String FIRSTOUTPUT = "input_array.txt";
    static final String SECONDOUTPUT = "tested_sorted_array.txt";
    Random rand = new Random();

    TestHarness(){
        setUpWriter(FIRSTOUTPUT);

        int[] randArray =
createRandomArray(rand.nextInt((int)Math.pow(2, 16)));
        output(randArray);

        endWriter();
        setUpWriter(SECONDOUTPUT);

        int[] sortedArray = SortingAlgorithm.SortArray(randArray);
        testArrayIsSorted(sortedArray);
        endWriter();
        return;
    }

    /**Checks if integer array is sorted from least to greatest
     *
     * @param array Array to check if it is sorted
     * @return  returns if the array was sorted
     */
    boolean testArrayIsSorted(int[] array){
        boolean errorFree = true;
        int errorCounter = 0;
        String[] errorLog = new String[50];

        for(int i = 0; i < array.length; i++)  //Iterates through
array and checks if each element is greater or equal to the previous
element
        {
            output(i+":  \t" + array[i]);

            if(i > 0 && array[i] < array[i-1])  //If the current
element is less than the previous element, the system will scream it
out in red
            {
                output(errorLog[errorCounter] = "\u001B[31m" +
"Element " + i + " (" + array[i] + ") is less than element " + (i-1)+
" (" + array[i-1] +   ")\u001B[0m");
                errorFree = false;
                errorCounter++;
```

```
            }
        }

        //Outputs the number of errors if there were any, and then
returns if there were any errors
        if(!errorFree)
        {
            System.out.println("\u001B[31m" + "\n\nThere was at least
" + errorCounter + " error(s). Please look for the locations in the
sorted array\n\n" + "\u001B[0m");
            for(int i = 0; i < errorCounter; i++)
System.out.println(errorLog[i]);
        }else
        {
            System.out.println("\u001B[32m" + "There were no errors"
+ "\u001B[0m");
        }
        return errorFree;
    }

    /**Creates a random array of inputted size to be run through the
sorting algorithm as a test.
     * each element of the array is a random integer of
     *
     * @param size
     * @return
     */
    int[] createRandomArray(int size){
        int[] temp = new int[size];
        for(int i = 0; i < size; i++) temp[i] = randomInt();
        return temp;
    }

    /**Creates a random integer of any possible integer value.
     *
     * @return  A random integer of any possible integer value
     */
    int randomInt(){
        int temp = rand.nextInt();
        if(rand.nextBoolean())  return temp;
        return -temp;
    }

    /**Sets up the file writer with the given file name
     *
     * @param fileName Name of the output file
     */
    void setUpWriter(String fileName) {
        try {
            output = new File(fileName);
            writer = new FileWriter(fileName);
        }catch (IOException e){
            System.out.println("error in creating the file");
        }
    }

    /**Closes the writer when you're done with it
     *
     */
    void endWriter(){
        try {
```

```
            writer.flush();
            writer.close();
        }catch (IOException e){
            System.out.println("The Writer must be used before it can
be ended");
        }
    }

    /**Outputs each element of given array to the output file
     * Each element gets its own line
     * Prints in array order (element 0, element 1, element 2, etc.,)
     * Only works on full arrays
     *
     * @param array
     */
    void output(int[] array){
        for(int i = 0; i < array.length; i++) {
            output(Integer.toString(array[i]));
        }
    }

    /**Outputs the inputted parameter to output file
     *
     * @param out output
     */
    void output(String out) {
        try {
            writer.write(out + "\n");
        }catch (IOException e){
            System.out.println("help im on fire");
        }
    }

    //does the runny programmy thing
    public static void main(String[] args) throws IOException
{TestHarness t = new TestHarness();}
}
```

My method works by creating an array of a random size (between 1 and 2^16 for time sake and heap space reasons,) of random ints that can be any value of integer with all integer values having an equal possibility of being added to the array (aside from 0, which can be called as either 0 or -0, giving it twice the chances of being called, but this won't impact anything). I then print each element of the array before I run the random array through the sorting algorithm. I then validate the array is sorted by comparing each element (starting at index 1,) to the previous element and logging if it is greater than the previous element.

```
There was at least 2 error(s). Please look for the locations in the sorted array


Element 1 (-2147195750) is less than element 0 (-569024030)
Element 7716 (-1579999150) is less than element 7715 (2145303077)
```

Through this I was able to determine that my sorting algorithm misses the first and last elements.

```
Arrays.sort(array, fromIndex: 1, toIndex: array.length-1)
return array;
```

The min index should be 0, (I expected this honestly,) but I didn't realize the library auto adjusts for the 0 index, so the ending index should be array.length.

To compile my code, just put the 2 classes into a single file, open that file in a java compiler, and run it. You can find this code on my GitHub.

B) Provide a context-free grammar to generate all the possible test-cases. **(18 + 8 = 26 pts)**

For each possible integer value

4- A) For the following code, manually draw a control flow graph to represent its logic and structure.

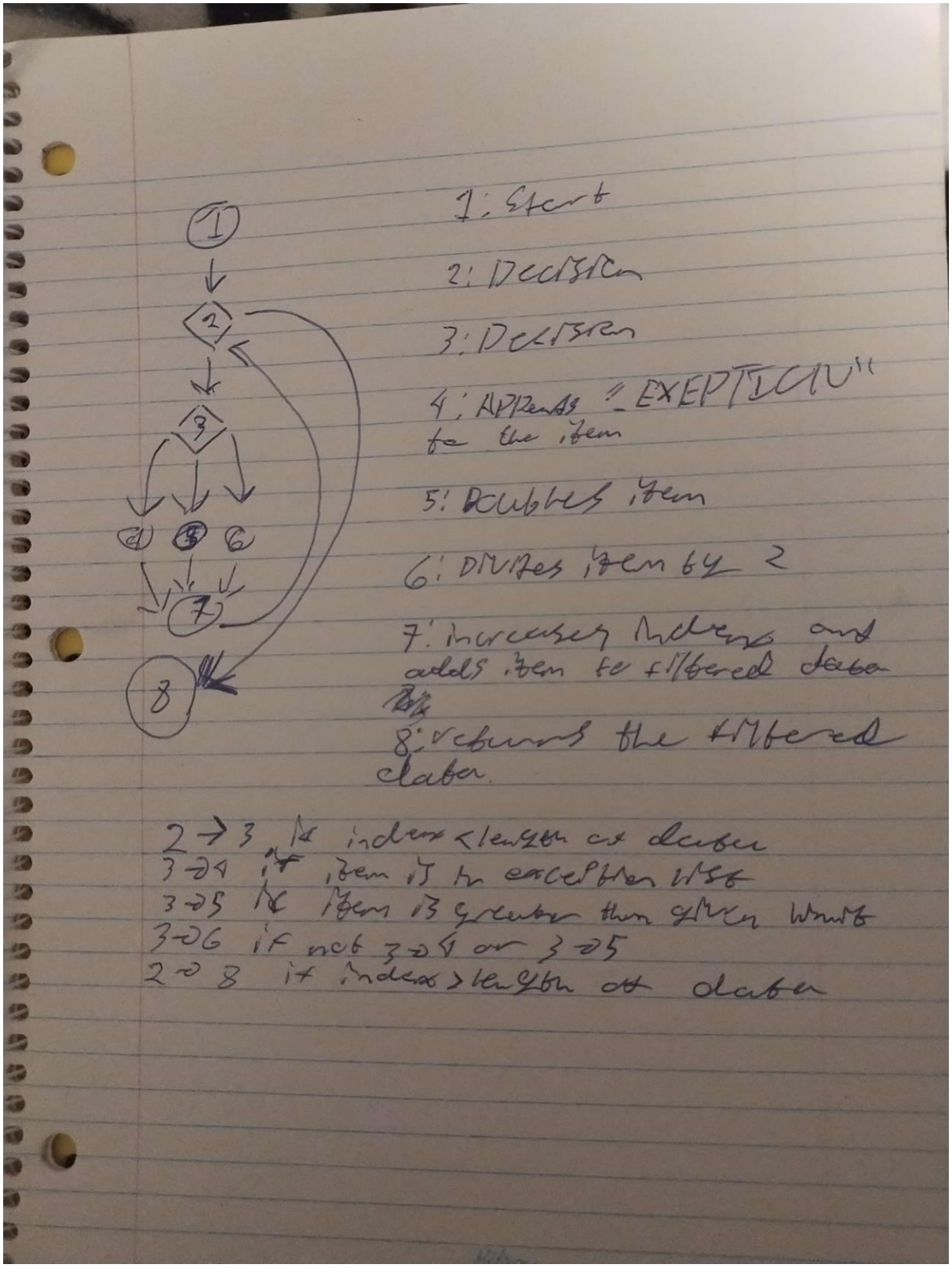
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

The code is supposed to perform the followings:

a. If an item is in the exceptions list, the function appends "_EXCEPTION" to the item.
b. If an item is greater than a given limit, the function doubles the item.
c. Otherwise, the function divides the item by 2.



B) Explain and provide detailed steps for “random testing” the above code. No need to run any code, just present the coding strategy or describe your testing method in detail. **(8 + 8 = 16 pts)**

For random testing of the above code, data should be of random size and length, alternating between valid arrays and invalid arrays, limit would be randomly inputted, both positive, negative, and as any

possible value for it’s type being a possible target, and the exceptions list should be of a random size with random contents.

5- A) Develop 4 distinct test cases to test the above code, with code coverage ranging from 30% to 100%. For each test-case calculate and mention its code coverage.

({1, 2, 3, 4, 5, 6}, -999, {2})
This covers over 66% of the code, since it will go both down the exception path, and down the over limit path.
({1, 2, 3, 4, 5, 6}, 4, {2, 3})
This covers 100% of the code, since it’ll test each path
({1, 2, 3, 4, 5, 6}, 7, {})
This covers 30% of the code since it’ll test each
({1, 2, 3}, 3, {2})
100% code coverage, since it'll test each path once.

B) Generate 6 modified (mutated) versions of the above code.

1.
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index > len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```
2.
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index == len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

3.

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item NOT in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1
 return filtered_data
```

4.

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item <= limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

5.

```
def filterData(data, exceptions):
    filtered_data = []
    index = 0
   limit = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
```

```
        index += 1

    return filtered_data
```

6.

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return exceptions
```

C) Assess the effectiveness of the test cases from part A by using mutation analysis in conjunction with the mutated codes from part B. Rank the test-cases and explain your answer.

D) Discuss how you would use path, branch, and statement static analysis to evaluate/analyse the above code. **(4 * 8 = 32 pts)**

To use path analysis, I would create a control flow graph, which would make it easy to understand.

To use branch analysis, I would run a bunch of test cases that hit each branch once. I would have 6 tests. If we treat a 1 as a parameter that would enter the specific branch if the others aren't entering, and 0 as the opposite, I would test the following: 000, 001, 010, 011, 100, 101, 110, 111.

To use statement analysis, I would look at the code and create 5 test cases that would have 0%, 100%, 30%, 30%, and 30% coverage respectively.

6- The code snippet below aims to switch uppercase characters to their lowercase counterparts and vice versa. Numeric characters are supposed to remain unchanged. The function contains at least one known bug that results in incorrect output for specific inputs.

```
def processString(input_str):
    output_str = ""
    for char in input_str:
        if char.isupper():
            output_str += char.lower()
        elif char.isnumeric():
            output_str += char * 2
        else:
```

```
output_str += char.upper()

return output_str
```

In this assignment, your tasks are:

a. Identify the bug(s) in the code. You can either manually review the code (a form of static analysis) or run it with diverse input values (a form of manual random testing). If you are unable to pinpoint the bug using these methods, you may utilize a random testing tool or implement random test case generator in code. Provide a detailed explanation of the bug, identify the line of code causing it, and describe your strategy for finding it.

The program will print any numeric value in a string twice, which is caused in the following code:

```
elif char.isnumeric():
    output_str += char * 2
```

The code runs bug free by simply removing the branch, or by making it

```
elif char.isnumeric():
    output_str += char
```

I found this by first doing some random testing on edge cases that will commonly not be programmed correctly (numerical values in strings, special chars, empty strings, etc.,) and once I found that numeric values within the strings get duplicated, I then looked through the code for possible causes. Basic branch analysis ruled out the first and second branches from being the culprit, since they only affect alphabetic values. A quick glance at the numeric branch showed that unnecessary operations were being done to the numeric value, and removing them solves the bug. However, knowing that Python's char.upper on a numeric value will return itself, lead me to realize the entire numeric branch is pointless as it will be covered by the else statement just fine.

b. Implement Delta Debugging, in your preferred programming language to minimize the input string that reveals the bug. Test your Delta Debugging code for the following input values provided.

```
/**Delta Debugs the UpperLower class to find the smallest substrings of
strings that are misbehaving for
 * UpperLower.processString
 *
 * @version 1.0 (Oct 16th, 2023)
 * @author Steve Mastrokalos   */

import java.io.IOException;

import static java.lang.Character.*;
import static java.lang.Character.toUpperCase;

public class DeltaDebug {
    String[] minValuesForBugs;  //String of the smallest bug-causing inputs
found
    int bugIndex = 0;           //Number of bugs found, bug array current
index
```

```
    static final String[] STRINGS_TO_TEST = {"abcdefG1", "CCDDEExy",
"1234567b", "8665"};

    /**Tests all given inputs for UpperLower.processString
     *
     */
    DeltaDebug(){
        for(int i = 0; i < STRINGS_TO_TEST.length; i++)
        {
            String temp = STRINGS_TO_TEST[i];
            if(!isBehavingAsIntended(temp)) test(temp);
            else System.out.println("no bugs found in string: " + temp);
            bugIndex = 0;
        }
    }

    /**Tests the UpperLower.processString on given string
     *
     * @param toTest    String to test UpperLower.processString with
     */
    void test(String toTest){

        vUtil.pageBreak();
        minValuesForBugs = new String[toTest.length()];
        deltaDebug(toTest);
        System.out.println(UpperLower.processString(toTest));
        System.out.println(UpperLower.expectedProcessedString(toTest));

        for(int i = 0; i < bugIndex; i++){
            System.out.println(minValuesForBugs[i]);
        }
        if(bugIndex == 0) System.out.println("no bugs found in string");
        System.out.println(vUtil.pageBreak(false));
    }

    /**Finds the smallest possible sub strings that result in bugs
     *
     * Looks if bugs occurre in any strings of size 1, then 2, then 3,
etc.,
     *
     * @param input String to find if it results in a bug when thrown
through UpperLower
     */
    void deltaDebug(String input){
        int minSize = 1;
        while(bugIndex == 0){
            deltaDebug(input, minSize);
            minSize++;
        }
    }

    /**Takes in a String array of size 2 or more and runs deltaDebug on the
first 2 strings in the array
     * For taking 2 halves of a string and deltaDebugging each half
     *
     * @param input Strings to be deltaDebugged
     * @param minSize   Minimum size for a bug to appear
     */
    void deltaDebug(String[] input, int minSize){
        deltaDebug(input[0], minSize);deltaDebug(input[1], minSize);
    }
```

```
    /**deltaDebugs inputted string, finds the smallest substrings that
result in unexpected behaviour
     *Recursively splits the string in half until the smallest parts with
bugs are found
     *
     * @param input     String to find bugs in
     * @param minSize   Minimum size to look for bugs in
     */
    void deltaDebug(String input, int minSize){
        if(isBehavingAsIntended(input)) return; //Backs up recursive search
if no bugs are in the remaining string
        if(input.length() < minSize) return;    //Backs up recursive search
if we are below the min size
        if(input.length() == minSize && !isBehavingAsIntended(input)){
//records any bugs found if at the minimum size
            minValuesForBugs[bugIndex] = input;
            bugIndex++;
            return;
        }
        else    //If we can split the string more, breaks it into 2 halves
and delta debugs each half
        {
            int delta = (int)Math.floor(input.length()/2);
            deltaDebug(halves(input), minSize);
        }
        return;
    }

    /**Splits a string into 2 halves, each put into a 2 element array
     * halves[0] = first half of the string
     * halves[1] =
     *
     * @param input
     * @return
     */
    String[] halves(String input){
        int delta = (int)Math.floor(input.length()/2);
        String front = "";
        String back = "";

        for(int i = 0; i < delta; i++)
        {
            front += input.charAt(i);
        }
        for(int i = delta; i < input.length(); i++){
            back +=input.charAt(i);
        }

        String[] array = {front, back};

        return array;
    }

    /**Returns a boolean value that the program is behaving as expected for
the given input
     * True if expected result given
     * False if not
     * @param input String to be run through UpperLower
     * @return  Boolean value of if UpperLower is giving the expected
result for the given input.
```

```
     */
    boolean isBehavingAsIntended(String input){

if(UpperLower.processString(input).equals(UpperLower.expectedProcessedStrin
g(input))) return true;
        return false;
    }



    //does the runny programmy thing
    public static void main(String[] args) throws IOException {DeltaDebug d
= new DeltaDebug();}
}

/**A program that flips upper and lowercases in a string
 *
 *@version 1.0 (Oct 16th, 2023)
 *@author Steve Mastrokalos   */

import static java.lang.Character.*;

public class UpperLower {
    public static String processString(String input_str) {
        String output_str = "";;
        for(int i =0; i < input_str.length(); i++) {
            char c = input_str.charAt(i);
            if (isUpperCase(c)) {
                output_str += toLowerCase(c);
            } else if (isDigit(c)) {
                output_str += c;
                output_str += c;
            } else {
                output_str += toUpperCase(c);
            }
        }
        return output_str;
    }

    public static String expectedProcessedString(String input_str) {
        String output_str = "";;
        for(int i =0; i < input_str.length(); i++) {
            char c = input_str.charAt(i);
            if (isUpperCase(c)) {
                output_str += toLowerCase(c);
            } else if (isDigit(c)) {
                output_str += c;
            } else {
                output_str += toUpperCase(c);
            }
        }
        return output_str;
    }
}



/**Personal utility class of mine
 * I cleaned it up so only the used stuff is in it
 *
```

```
 *@version 2.0 (Oct 16th, 2023)
 *@author Steve Mastrokalos   */

public class vUtil {

    /**Creates a line of astriks to create a clean pagebreak
     *
     * @return
     */
    public static String pageBreak(){return pageBreak(true);}

    /**Creates a line of astriks to create a clean pagebreak
     *
     * @param Print Whether or not to print the line of astrisks
     * @return  line of astrisks
     */
    public static String pageBreak(boolean Print){
        String out = "";
        for(int i = 0; i < 30; i++) out += "*";

        if(Print) System.out.println("\n" + out + "\n");

        return out;
    }
}
```

i. “abcdefG1”

{1}

ii. “CCDDEExy”

{}

iii. “1234567b”

{1, 2, 3, 4, 5, 6, 7}

iv. “8665”

{8, 6, 6, 5}

Briefly explain your delta-debugging algorithm and its implementation and provide the source code in/with your assignment. **(4 + 12 = 16 pts)**

My delta debugging algorithm take a string input that causes bugs, and recursively halves it until all of the smallest bug causing strings are found. If the input string causes no bugs, my algorithm will say that before it starts.

7- Extra Credit Assignment: Create a GitHub repository to host all the elements of this assignment. This includes source codes, test data, and any screenshots or logs you have generated. Submit the GitHub link along with your main submission through Brightspace. **(5 pts)**

https://github.com/vestex727/COSC-3P95

File Assignment 1

Marking Scheme:

Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Lack of clarity may lead you to lose marks, so keep it simple and clear.

Submission:

The submission is expected to contain a sole word-processed document. The document can be in either ***DOC or PDF*** *format; it should be a single column, at least single-spaced, and at least in font 11. It is strongly recommended to use the assignment questions to facilitate marking: answer the questions just below them for easier future reference.*

Late Assignment Policy:

A one-time penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, four days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion.

Plagiarism:

Students are expected to respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee's attention. In case plagiarism is determined, the activity will be canceled, and the author(s) will be subject to university regulations. For further information on this sensitive subject, please refer to the document below: ***https://brocku.ca/node/10909***