Homework 3 - Implementing a *Hash* function

COP3503 Assigned: April 21, 2021
Michael McAlpin, Instructor Due: May 1, 2021

1 Objective

Build a hashing algorithm that is suitable for use in a Bloom Filter. Please note that while a cryptographic hash is quite common in many Bloom Filters, the hashing algorithm to be implemented is a mix of the the following algorithmic models, specifically, a *multiply & rotate* hash colloquially known as a *murmur* hash, and an *AND, rotate, & XOR* hash colloquially known as an *ARX* hash.

2 Requirements

- Inputs.
    - Read the input file which contains strings of characters, with **one** string per line. Please note that the strings may vary in size from a single character to many characters, perhaps as long as 35 characters and may be possibly longer.
        - All the input characters will be in 8 bit ASCII not **UTF**.
    - Calculate the hash value for the single string in the particular line under test.

2.1 Algorithm Design

The pseudocode for the ***UCFxram*** hashing algorithm is shown below. Please note the following:

- The input string is in ASCII in either a Java string or as an array of characters in bytes.
- Once the the single word has been read, the ***UCFxram*** function calculates the 32 bit integer hash value derived from the input text.
- Upon completion of the hash calculation, the hash value is output as specified in the **Outputs** section.

Algorithm 1 UCFxram(String input, int len)

```
$randVal1 \leftarrow 0xbcde98ef$          // aribtrary value
$randVal2 \leftarrow 0x7890face$
$hashVal \leftarrow 0xfa01bc96$          // start seed value
$roundedEnd \leftarrow len \ \& \ 0xfffffffc$          // Array d gets 4 byte blocks
for $i = 0; i < roundedEnd; i + 4$ do
    $tempData \leftarrow (data[i] \ \& \ 0xff) \ | \ ((data[i+1] \ \& \ 0xff) \ << \ 8) \ | \ ((data[i+2] \ \& \ 0xff) << 16) \ | \ (data[i+3] \ << \ 24$
    $tempdata \leftarrow tempData \ * \ randVal1$          // Mulitiply
    $ROTL(tempData, 12)$          // Rotate left 12 bits
    $tempdata \leftarrow tempData \ * \ randVal2$          // Another Multiply
    $hashValue \leftarrow hashValue \ \oplus \ tempData$
    $ROTL(hashValue, 13)$          // Rotate left 13 bits
    $hashValue \leftarrow hashValue \ * \ 5 \ + \ 0x46b6456e$
            // Now collect orphan input characters
$tempData \leftarrow 0$
if $(len \ \& \ 0x03) == 3)$ then
    $tempData \leftarrow (data[roundedEnd \ + \ 2] \ \& \ 0xff) << 16$
    $len \leftarrow len \ - \ 1$
if $(len \ \& \ 0x03) == 2)$ then
    $tempData \ |= \ (data[roundedEnd \ + \ 1] \ \& \ 0xff) << 8$
    $len \leftarrow len \ - \ 1$
if $(len \ \& \ 0x03) \ == \ 1)$ then
    $tempData \ |= \ (data[roundedEnd] \ \& \ 0xff)$
    $tempData \leftarrow tempData \ * \ randVal1$          // Mulitiply
    $ROTL(tempData, 14)$          // Rotate left 14 bits
    $tempData \leftarrow tempData \ * \ randVal2$          // Another Multiply
    $hashValue \leftarrow hashValue \ \oplus \ tempData$
$hashValue \leftarrow hashValue \ \oplus \ len$          // XOR
$hashValue \leftarrow hashValue \ \& \ 0xb6acbe58$          // AND
$hashValue \leftarrow hashValue \ \oplus \ hashValue >>> 13$
$hashValue \leftarrow hashValue \ * \ 0x53ea2b2c$          // Another arbitrary value
$hashValue \leftarrow hashValue \ \oplus \ hashValue >>> 16$
RETURN hashValue          // Return the 32 bit int hash
```

2.1.1 Outputs

The output of the ***UCFxram*** hashing algorithm is deceptively simple and shown below.

Hash Value `System.out.format("%10x:%s\n",hashValue, input );`

Completion `System.out.println("Input file processed");`

2.1.2 Functions

complexityIndicator Prints to **STDERR** the following:

NID

A difficulty rating of difficult you found this assignment on a scale of 1.0 (easy-peasy) through 5.0 (knuckle busting degree of difficulty).

Duration, in hours, of the time you spent on this assignment.

Sample output:

```
ff210377@eustis:~/COP3503$ ff210377;3.5;18.5
```

3 Testing

Make sure to test your code on Eustis **even if it works perfectly on your machine**. If your code does not compile on Eustis you will receive a 0 for the assignment. There will be 10 input files and 9 output files provided for testing your code, they are respectively shown in Table 1 and in Table 2.

Filename	Description
oneA.txt	A single upper case A
twoAs.txt	Two upper case As
threeAs.txt	Three upper case As
fourAs.txt	Four upper case As
5words.txt	5 words
100words.txt	100 words
500words.txt	500 words
1000words.txt	1000 words
2500words.txt	2500 words

Table 1: Input files

The expected output for these test cases will also be provided as defined in Table 2. To compare your output to the expected output you will first need to redirect *STDOUT* to a text file. Run your code with the following command (substitute the actual names of the input and output file appropriately):

```
java Hw03 inputFile > output.txt
```

The run the following command (substitute the actual name of the expected output file):

```
diff output.txt expectedOutputFile
```

If there are any differences the relevant lines will be displayed (note that even a single extra space will cause a difference to be detected). If nothing is displayed, then congratulations - the outputs match! For each of the five (5) test cases, your code will need to output to *STDOUT* text that is identical to the corresponding *expectedOutputFile*. If your code crashes for a particular test case, you will not get credit for that case.

4 Submission - via WebCourses

The ***single*** Java source file, named **Hw03.java**. Make sure that the *main* program is in Hw03.java. Use reasonable and customary naming conventions for any classes you may create for this assignment.

5 Sample output

```
ff210377@eustis:~/cop3503/hw3 $ java Hw03 5words.txt
ff210377;3.5;18.5
  25e69846:aaaa
  c3bef462:bbbbbbbbb
  c6e29ee2:ccccccccddddddeeeeffff
  d633fb53:bananas
  7a8fa153:syzygy
Input file processed
ff210377@eustis:~/cop3503/hw3 $java Hw03 5words.txt >5wordsSt.txt
ff210377;3.5;18.5
ff210377@eustis:~/cop3503/hw3 $ diff 5wordsSt.txt 5wordsValid.txt
ff210377@eustis:~/cop3503/hw3 $
```

*Note: This is based on an actual **UCFxram** output using the inputs specified in the commands shown above.*

Note The **ff210377;3.5;18.5** output shown above is the output from the *complexityIndicator* function to **STDERR**.

Command	Validly formatted output files
java Hw03 oneA.txt > oneASt.txt	oneAValid.txt
java Hw03 twoAs.txt > twoAsSt.txt	twoAsValid.txt
java Hw03 threeAs.txt > threeAsSt.txt	threeAsValid.txt
java Hw03 fourAs.txt > fourAsSt.txt	fourAsValid.txt
java Hw03 5words.txt > 5wordsSt.txt	5wordsValid.txt
java Hw03 100words.txt > 100wordsSt.txt	100wordsValid.txt
java Hw03 500words.txt > 500wordsSt.txt	500wordsValid.txt
java Hw03 1000words.txt > 1000wordsSt.txt	1000wordsValid.txt
java Hw03 2500words.txt > 2500wordsSt.txt	2500wordsValid.txt

Table 2: Commands with input files and corresponding output files.

6 Grading

Grading will be based on the following rubric:

Percentage	Description
-100	Cannot compile on *Eustis*.
-100	Cannot read input files.
- 50	Does not produce valid and exact hash value for small input files.
- 25	Cannot produce exact hash values for large input files.
- 15	Hash value product off by less than 3 bits.
- 10	Output does not match *expectedOutput.txt* exactly. (*Whitespaces will be ignored.*)

Table 3: Grading Rubric