CT255 Cybersecurity

Assignment 1

Maxwell Maia 21236277

Problem 2

```
import java.util.Random;

/**
 *
 * @author Maxwell Maia 21236277, built on top of code supplied by Michael Schukat
 */
public class CT255_HashFunction1 {

    public static void main(String[] args) {

        System.out.println("");
        System.out.println("");
        System.out.println("");

        int res = 0;

        if (args != null && args.length > 0) { // Check for <input> value
            res = hashF1(args[0]); // call hash function with <input>
            if (res < 0) { // Error
                System.out.println("Error: <input> must be 1 to 64 characters long.");
            }
```

```
        else {
            System.out.println("input = " + args[0] + " : Hash = " + res);
            System.out.println("");
            System.out.println("Start searching for collisions with "+"\""+args[0]+"\".");
            System.out.println("");

            // Your code to look for a hash collision starts here!

            int count = 0; //Integer to count collisions
            String[] collisions = new String[10]; //String array to store successul collisions.
            int tempHash = 0; //Integer to temporarily store a hash.

            //Loop until 10 collisions are found...
            while(count < 10)
            {
                String temp = randomString(); //Generate a random string of 10 characters
                tempHash = hashF1(temp); //hash the random string

                if(tempHash == res) //if the hash of the random string matches the hash of the
target hash, then a collision has been found.
                {
                    collisions[count] = temp; //save the random string input that caused a
collision.
                    count++; //Increment amount of collisions found.
                }
            }

           //Print out all 10 collisions.
            for(int k = 0; k < 10; k++)
            {
                System.out.println("Collision "+ (k+1) + ": " + collisions[k]);
            }
        }
```

```
        }
        else { // No <input>
            System.out.println("Use: CT255_HashFunction1 <Input>");
        }
    }

    //Method that returns a string of length 10 of random characters.
    public static String randomString()
    {
        String randomString = "";
        Random random = new Random();

        for(int j = 0; j < 10; j++)
        {
            char randomCharacter = (char) (random.nextInt(26) + 'a'); //generate a random
character
            randomString += randomCharacter; //add it to the string
        }

        return randomString;
    }


    private static int hashF1(String s){
        int ret = -1, i;
        int[] hashA = new int[]{1, 1, 1, 1};

        String filler, sIn;

        filler = new
String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEF
GHABCDEFGH");
```

```
        if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length
            ret = -1;
        }
        else {
            sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
            sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
            // System.out.println(sIn); // FYI
            for (i = 0; i < sIn.length(); i++){
                char byPos = sIn.charAt(i); // get i'th character
                hashA[0] += (byPos * 17); // Note: A += B means A = A + B
                hashA[1] += (byPos * 31);
                hashA[2] += (byPos * 101);
                hashA[3] += (byPos * 79);
            }

            hashA[0] %= 255;  // % is the modulus operation, i.e. division with rest
            hashA[1] %= 255;
            hashA[2] %= 255;
            hashA[3] %= 255;

            ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 *
256);
            if (ret < 0) ret *= -1;
        }
        return ret;
    }
}
```

```
input = Bamb0 : Hash = 1079524045

Start searching for collisions with "Bamb0".

Collision 1: ahkcanfagb
Collision 2: cvfabfceae
Collision 3: ahhbracbef
Collision 4: gfeacampaa
Collision 5: fhcjaalded
Collision 6: cefabdhfne
Collision 7: dmcbbkadlb
Collision 8: obabighagb
Collision 9: jgcaahbang
Collision 10: eflafclccc
```

Problem 3

I upgraded the hash function. Added another index to the hashA integer array.

NEW HASH FUNCTION

```
private static int hashF1(String s){
    int ret = -1, i;
    int[] hashA = new int[]{1, 1, 1, 1, 1};
    // ADDED an extra index to integer array to make the hash more robust.
    //  This gives more combinations which means there is a lower possibily of a collision occurring.

    String filler, sIn;
```

```
    filler = new
String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEF
GHABCDEFGH");


    if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length
       ret = -1;
    }
    else {
       sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
       sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
       // System.out.println(sIn); // FYI
       for (i = 0; i < sIn.length(); i++){
          char byPos = sIn.charAt(i); // get i'th character
          hashA[0] += (byPos * 17); // Note: A += B means A = A + B
          hashA[1] += (byPos * 31);
          hashA[2] += (byPos * 101);
          hashA[3] += (byPos * 79);
          hashA[4] += (byPos * 48); // ADDED to make the hash more robust
       }


       hashA[0] %= 255;  // % is the modulus operation, i.e. division with rest
       hashA[1] %= 255;
       hashA[2] %= 255;
       hashA[3] %= 255;
       hashA[4] %= 255; // ADDED to make the hash more robust


       // ADDED + hashA[4] * 256 * 256 * 256 * 256)
       ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 *
256) + (hashA[4] * 256 * 256 * 256 * 256);
       if (ret < 0) ret *= -1;
    }
    return ret;
  }
```

```
input = hope : Hash = 1782428929

Start searching for collisions with "hope".

Collision 1: dgkkxjupth
Collision 2: pujvvakjmf
Collision 3: qhambwveqx
Collision 4: ekmjralzwm
Collision 5: uvhsqpjgge
Collision 6: sndqdyiiqn
Collision 7: rzceysqdlc
Collision 8: pprueugofg
Collision 9: ytwudrfdch
Collision 10: sontdnogll
```