Source code for problem 1 and 2

```
/*  CT255 Assignment 2
 *  This class provides functionality to build rainbow tables (with a different reduction function per round) for 8 character long strings, which
    consist of the symbols "a .. z", "A .. Z", "0 .. 9", "!" and "#" (64 symbols in total).
    Properly used, it creates the following value pairs (start value - end value) after 10,000 iterations of hashFunction() and reductionFunction():
          start value  -  end value
          Kermit12        lsXcRAuN
          Modulus!        L2rEsY8h
          Pigtail1        R0NoLf0w
          GalwayNo        9PZjwF5c
          Trumpets        !oeHRZpK
          HelloPat        dkMPG7!U
          pinky##!        eDx58HRq
          01!19!56        vJ90ePjV
          aaaaaaaa        rLtVvpQS
          036abgH#        klQ6IeQJ


 *
 * @author Michael Schukat
 * @version 1.0
 */

//@author Maxwell Maia, 21236277
public class RainbowTable
{
    /**
 * Constructor, not needed for this assignment
 */
public RainbowTable() {

}

public static void main(String[] args) {
    long res = 0;
    int i;
    String start;

    if (args != null && args.length > 0) { // Check for <input> value
        start = args[0];

        if (start.length() != 8) {
            System.out.println("Input " + start + " must be 8 characters long - Exit");
        }
        else {
            // Your code for problem 1 starts here
            // "String start" has the first word of chain

            //Declare variables
            String plaintext = start;
            long ciphertext = 0L;

            //Array of hash values to get passwords for (For problem 2)
            long[] hashInputArray = {895210601874431214L,
750105908431234638L, 111111111115664932L, 977984261343652499L};

            //Generate chain starting from the first word
            for(i = 0; i < 10000; i++)
```

```
          {
             //hash
             ciphertext = hashFunction(plaintext);

             //RETRIEVE PASSWORD USING HASH IN THIS CHAIN
             // For each hash in the hash input array, check whether is matches
one of the hashs
             for(int k = 0; k < hashInputArray.length; k++)
             {
                if(ciphertext == hashInputArray[k])
                {
                   //Hash match found. return the password
                   System.out.println("\n\n===Match found===");
                   System.out.println("Hash input: " + ciphertext);
                   System.out.println("Password found: " + plaintext);
                }
             }

             //reduce
             plaintext = reductionFunction(ciphertext, i);

          }

          //Print start and end
          System.out.println("\nChain");
          System.out.println("start value: " + start);
          System.out.println("end value: " + plaintext);
          System.out.println("\n=================\n\n");
       }
    }
    else { // No <input>
```

```
            System.out.println("Use: RainbowTable <Input>");
        }
    }

    private static long hashFunction(String s){
        long ret = 0;
        int i;
        long[] hashA = new long[]{1, 1, 1, 1};

        String filler, sIn;

        int DIV = 65536;

        filler = new
String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHA
BCDEFGHABCDEFGH");

        sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
        sIn = sIn.substring(0, 64); // // Limit string to first 64 characters

        for (i = 0; i < sIn.length(); i++) {
            char byPos = sIn.charAt(i); // get i'th character
            hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
            hashA[1] += (hashA[0] + byPos * 31349);
            hashA[2] += (hashA[1] - byPos * 101302);
            hashA[3] += (byPos * 79001);
        }

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


    private static String reductionFunction(long val, int round) {  // Note that for the first function call "round" has to be 0,
        String car, out;                                             // and has to be incremented by one with every subsequent call.
        int i;                                                       // I.e. "round" created variations of the reduction function.
        char dat;


        car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#");
        out = new String("");


        for (i = 0; i < 8; i++) {
            val -= round;
            dat = (char) (val % 63);
            val = val / 83;
            out = out + car.charAt(dat);
        }


        return out;
    }
}
```