TextHasher (1.01)

Kcits970

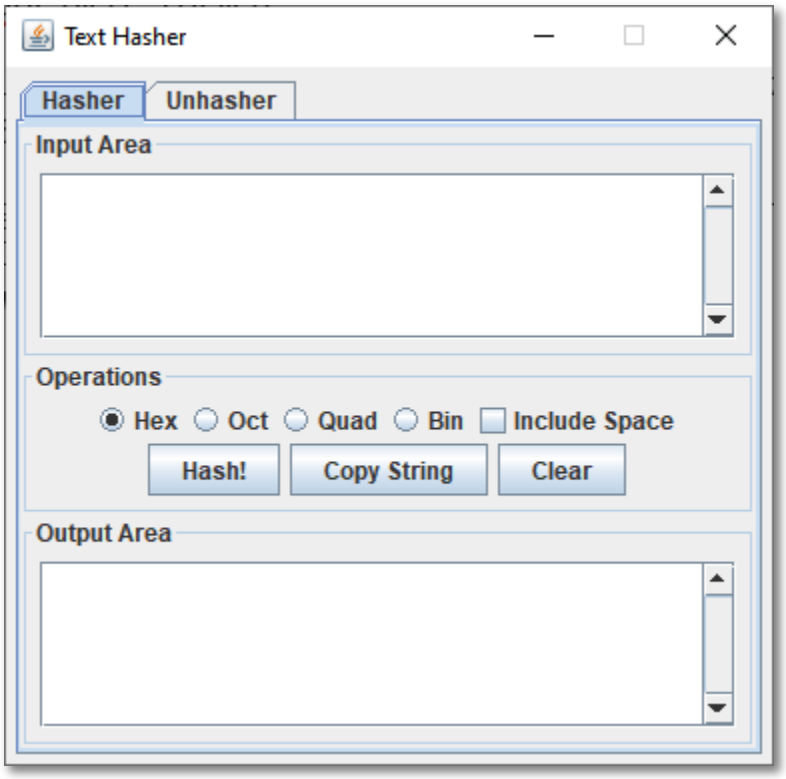
January 16, 2022

Sections

Section 1. Description

TextHasher is an event-driven interface to hash ASCII text. In this program, the user can hash text into binary, quaternary, octal, decimal, hexadecimal, and triacontakaidecimal format.

The initial version of TextHasher was created back in December 2020.



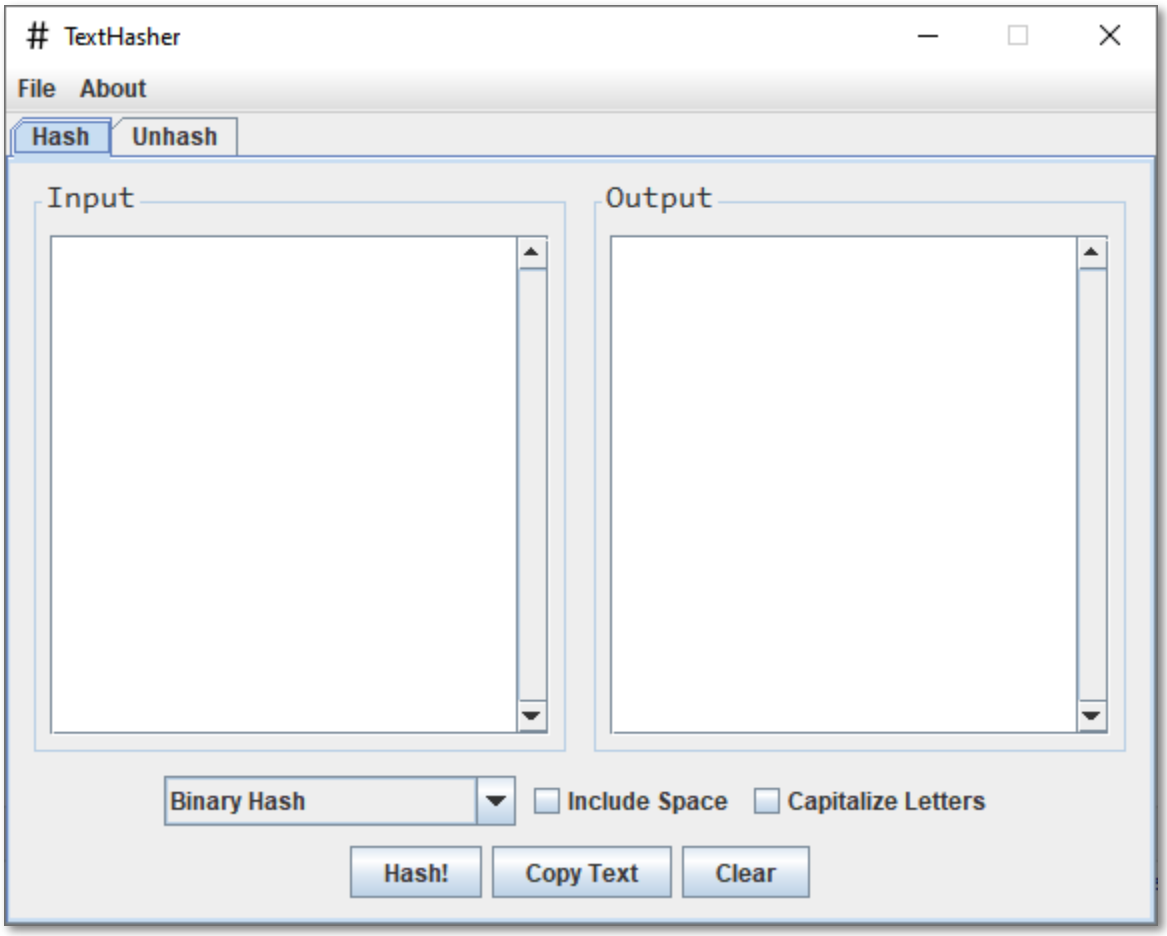
(Appearance of the initial interface)

Version 1.01 simply provides refactored code and overall improved GUI appearance*(Appearance of the new GUI is shown in the next section)*.

The program is written in Java language, and compiled using Java Compiler 15.0.1. Attempts to compile the given source code with a different compiler may result in unexpected or undefined behavior.

Section 2. Features

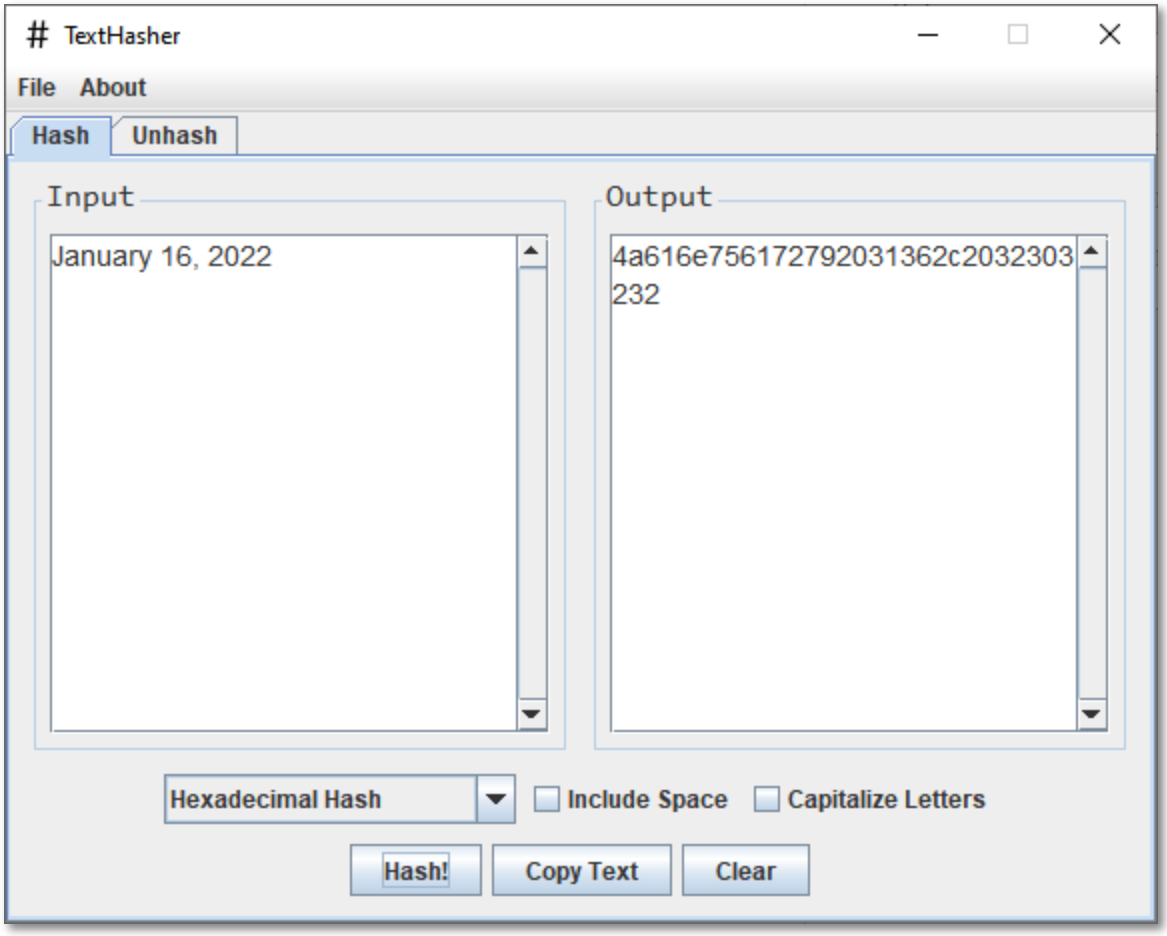
When the program is launched, an empty frame will be shown.



(The initial frame at program launch.)

Hashing Text

To hash any text, simply type in the text in the [Input] text area, select the desired hash function from the combo box, and click [Hash!].



(The frame after hashing "January 16, 2022" with hexadecimal hash.)

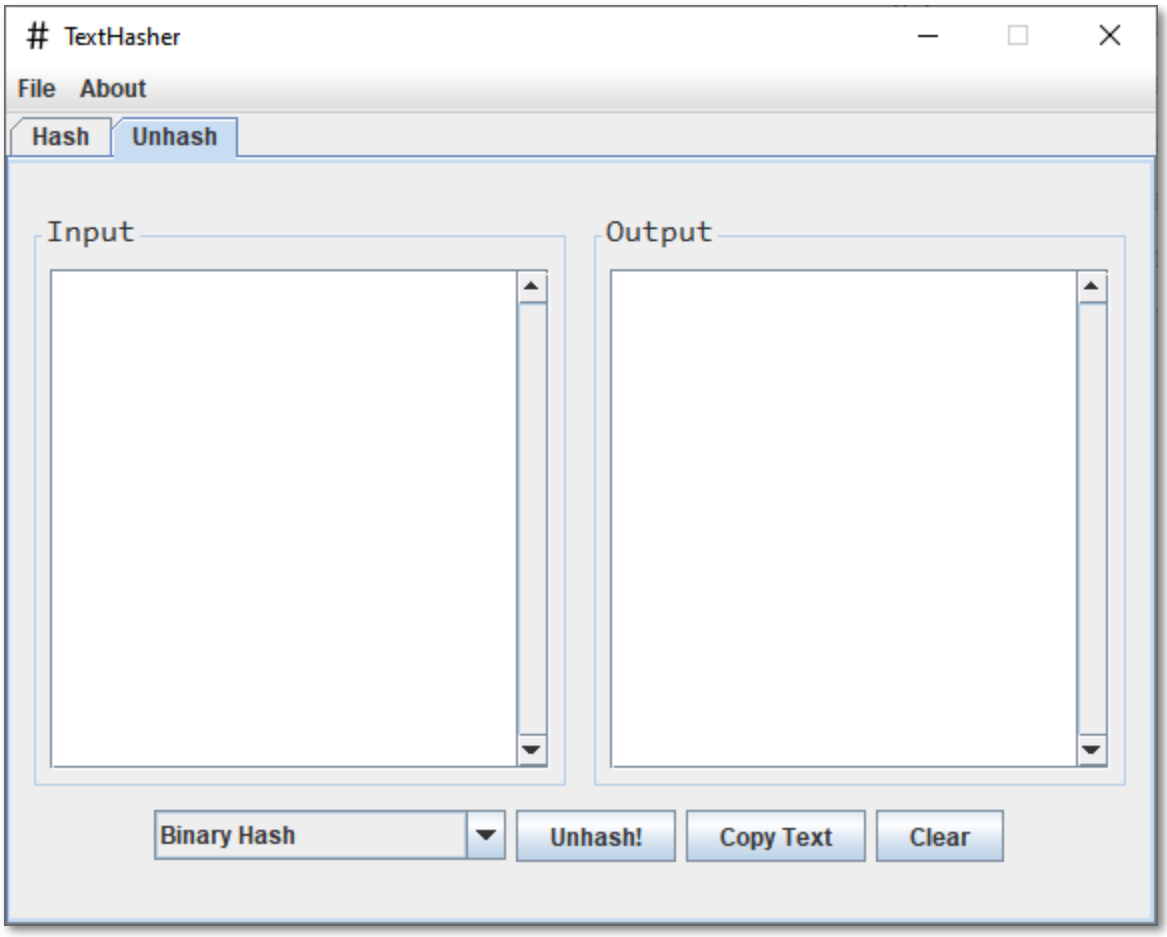
There are two checkboxes*("Include Space", "Capitalize Letters")* that change the appearance of the output hash string. The first checkbox includes a space character between the two hash units, and the second checkbox capitalizes the alphabet characters, if there are any. Below is an example of how different combinations of checkboxes affect the output hash.

<Output Format>

Include Space	***Capitalize Letters***	***Hash Function***	***Input***	***Output***
N	N	Hexadecimal	Kcits970	4b63697473393730
Y	N			4b 63 69 74 73 39 37 30
N	Y			4B63697473393730
Y	Y			4B 63 69 74 73 39 37 30

Unhashing Text

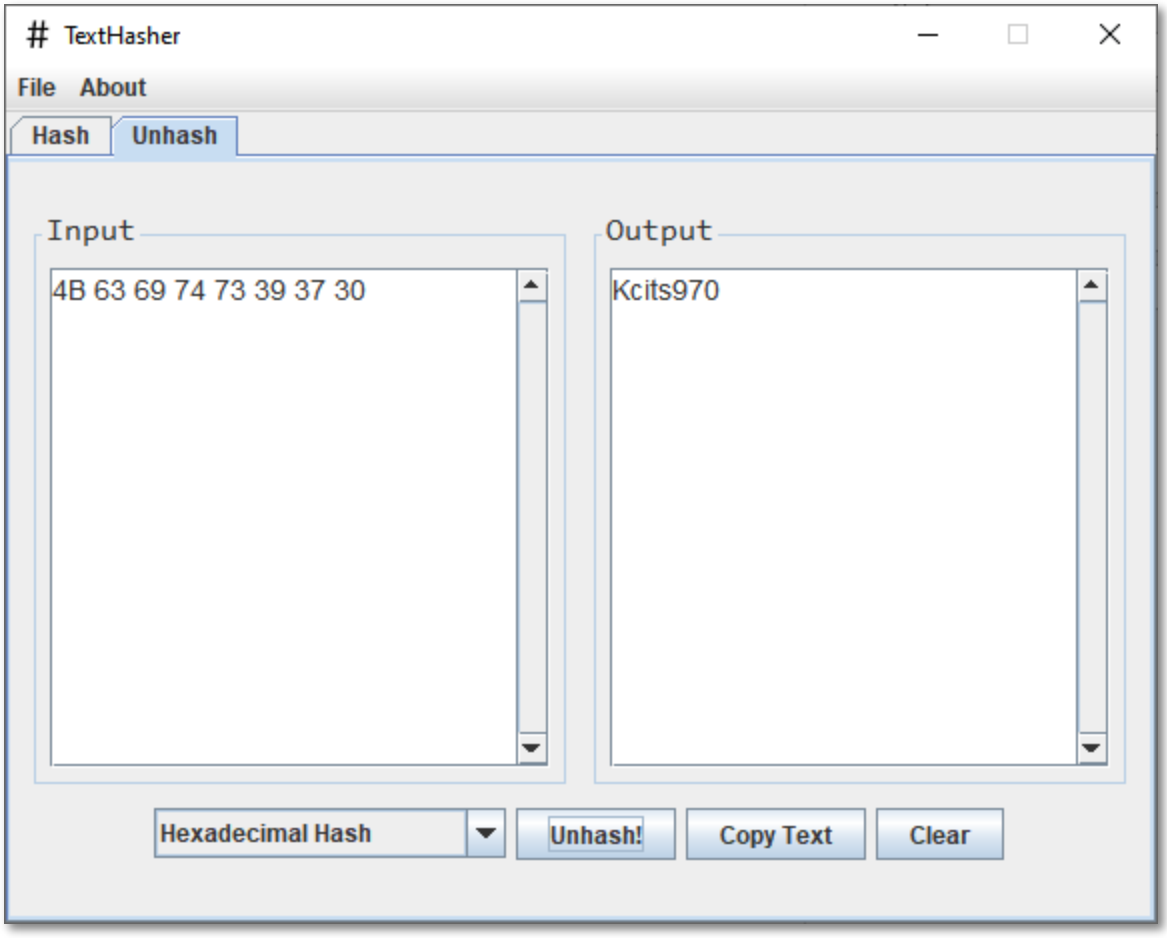
To unhash[1] the string back to its original state, switch to the [Unhash] tab.



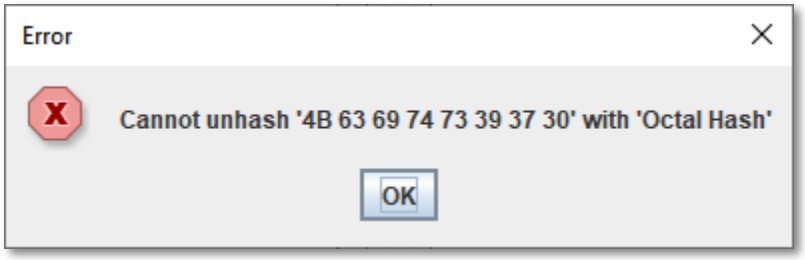
(The [Unhash] Tab)

Paste in the hashed text, select the hash function from the combo box, and click [Unhash!]. If unhashed successfully, the output text area will display the original string. Otherwise, an error dialog will pop up, and the output text area will display "NULL".

[1] 'Unhashing' is equivalent to parsing a hashed text.



(The frame after successfully unhashing the given string.)



(The Error Dialog)

Section 3. Hash Algorithm

TextHasher uses a relatively simple hash algorithm. The program takes each character of the string and converts it to an integer of a specific base. Each integer is then concatenated into one string, which completes the hash. To explain the process in further detail, I'll begin by explaining some basic terminology.

Base Terminology

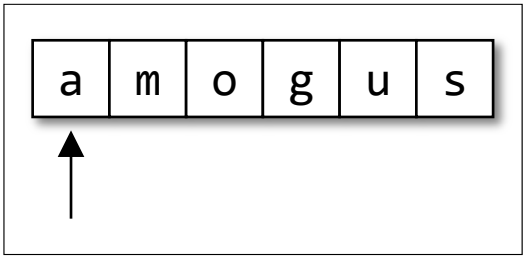
Hash Radix: The radix of the character code.
Hash Unit: A hash of a single character of the string.
Hash Unit Size: The length of the hash unit.
Hash Function: A function that returns the hash of a given string. A hash function is defined by 2 variables: hash radix and hash unit size.

The program provides 6 different hash functions:

Function Name	Hash Radix	Hash Unit Size
Binary Hash	2	8
Quaternary Hash	4	4
Octal Hash	8	3
Decimal Hash	10	3
Hexadecimal Hash	16	2
Triacontakaidecimal Hash	32	2

Hash Process

I'll take the string "amogus" and hash it using binary hash. I start by taking the first character of the string, 'a'.



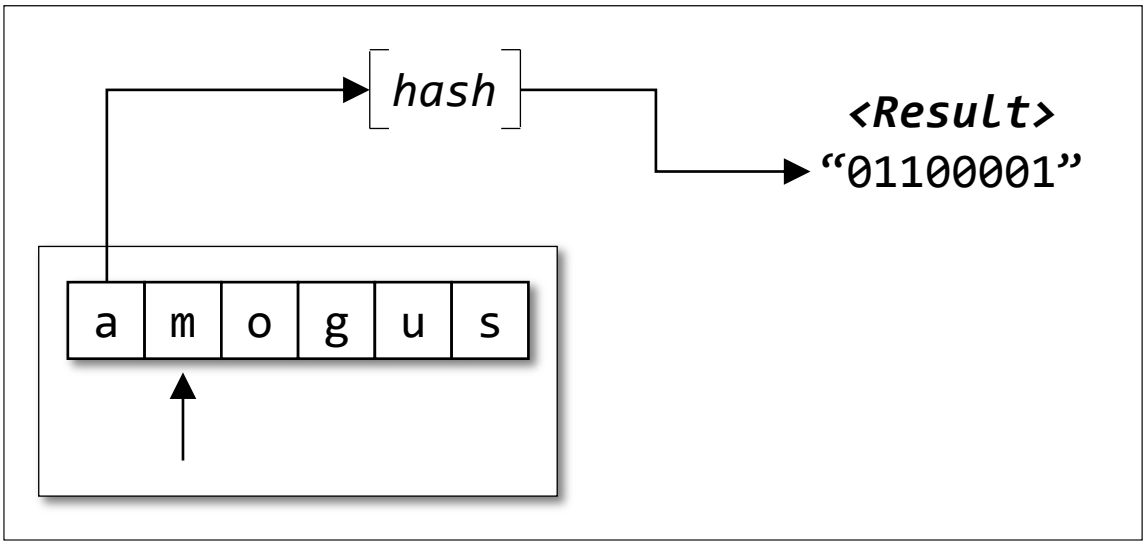
I see that the ASCII code of 'a' is equal to 97_{10}, but because we're using binary hash, I use the binary representation instead.

$$97_{10} = 1100001_2$$

In this case, “1100001” is the hash unit of ‘a’. Notice how the length of “1100001” is smaller than the hash unit size of binary hash[2]. To compensate for this offset, I pad the string with leading zeros.

$$\text{“1100001”} \rightarrow \text{“0}1100001\text{”}$$

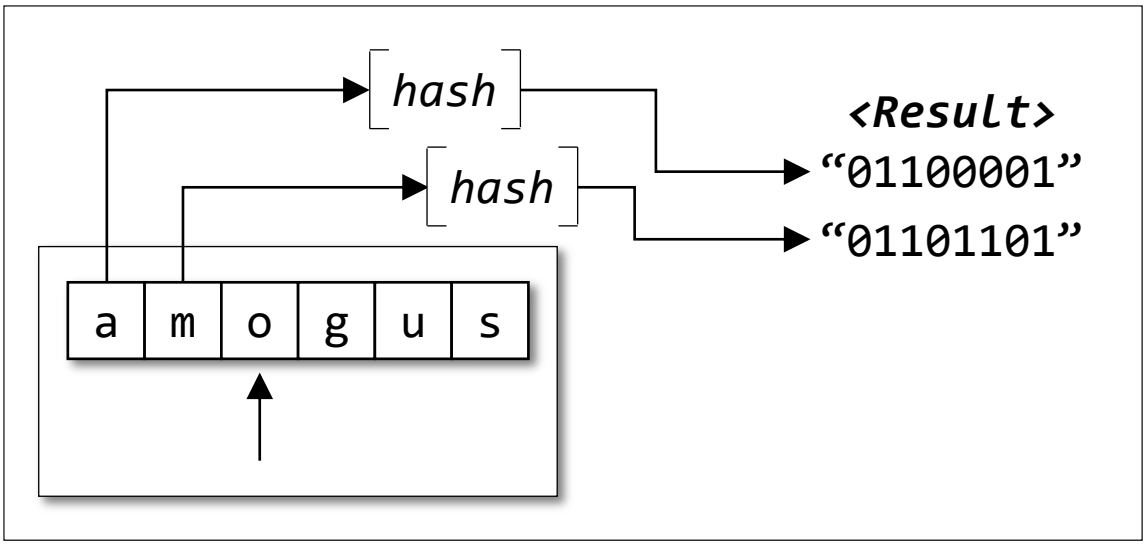
This completes the hash of the first character. I concatenate the hashed character (which is a string) to the result, and move on to the second character.



I follow the same process: converting to binary code, padding leading zeros if needed, and concatenating to the resulting hash string.

$$\text{ASCII(‘m’)} = 109_{10} = 1101101_2 \rightarrow \text{“0}1101101\text{”}$$

[2] “1100001” contains 7 characters, which is less than 8.



I continue this process until all of the characters are hashed. The complete binary hash of “amogus” is “01100001 01101101 01101111 01100111 01110101 01110011”[3].

Unhash Process

The process of unhashing is exactly the opposite of hashing. I’ll use the string “01110011 01110101 01110011” and attempt to unhash with binary hash to demonstrate the process. To start, I take the first 8 characters of the string[4].

“01110011 01110101 01110011”

Then I simply revert the string back to the original character.

$$01110011_2 = 115_{10} = \text{ASCII}(\text{‘s’}) \rightarrow \text{‘s’}$$

I continue the process until I reach the end of the string.

$$01110101_2 = 117_{10} = \text{ASCII}(\text{‘u’}) \rightarrow \text{‘u’}$$
$$01110011_2 = 115_{10} = \text{ASCII}(\text{‘s’}) \rightarrow \text{‘s’}$$

The finished binary unhash of “01110011 01110101 01110011” is “sus”.

[3] Without spacing, the hash is “011000010110110101101111011001110111010101110011”.
[4] This is because the size of a binary hash unit is 8. If I were to use octal hash, I would take the first 3 characters.