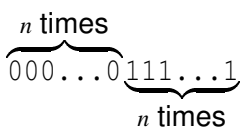
Construct a regular expression to match the following...

1. All bit patterns (`0`s and `1`s) of a 32 bit integer,

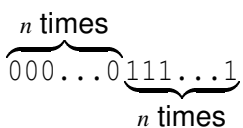
2. All $32b$ integer bit patterns **without** two sequential identically valued bits,

3. Bit representations of arbitrarily large integers conforming to the pattern



Construct a regular expression to match the following...

1. All bit patterns (`0`s and `1`s) of a 32 bit integer,
$$(0|1)^{32}$$
2. All $32b$ integer bit patterns **without** two sequential identically valued bits,
$$(01)^{16}|(10)^{16}$$
NOT $(01|10)^{16}$ WHICH PERMITS `0110...`
3. Bit representations of arbitrarily large integers conforming to the pattern



Construct a regular expression to match the following...

1. All bit patterns (`0`s and `1`s) of a 32 bit integer,

$$(0|1)^{32}$$

2. All $32b$ integer bit patterns **without** two sequential identically valued bits,

$$(01)^{16}|(10)^{16}$$

NOT $(01|10)^{16}$ WHICH PERMITS `0110...`

3. Bit representations of arbitrarily large integers conforming to the pattern

$\overbrace{000\ldots0}^{n \text{ times}}\underbrace{111\ldots1}_{n \text{ times}}$

0^n1^n ???

Limitations of Regular Languages

Regular languages cannot express arbitrarily deep recursive (nested) structures.

Examples:

- Cannot confirm matching parenthesis of mathematical expressions,
- Cannot confirm matching parenthesis of conventional compositional syntax of function calls (`log(sin(3))`),
- Cannot confirm matching curly braces of compound statements in C, C++, Java, Javascript, ...
- And the simplist example: $0^n 1^n$

For this very reasonable language feature, we need a higher level of language definitions: **context free grammars**.