LawaUnpa

Borja Martinena

March 2022

Contents

1 Introduction

LawaUnpa is a Brainfuck-like language, created for a CTF turned programming contest. It features syntax fully in Toki Pona, a minimalist conlang created in 2001 by Sonja Lang, but it's intention is not for the source code to be understandable as a Toki Pona text. My intention with LawaUnpa is to have a simple exercise, making an interpreter for it, that includes a bit of most of the basic features you'd expect to see in an imperative programming languaje: control-flow, data-types, data-structures, procedures/functions and IO. In this way, making a LawaUnpa interpreter makes for a good starting exercise when learning a new language.

LawaUnpa has and may go though some updates where I add new keywords for functionality that I find interesting, cool, weird, or simply feel like implementing, though my intention is keeping the language super simple and (I assume, based on the non infinite cell list) not Turing complete.

2 Structure

LawaUnpa is an extremely simple language that consisted of only 9 keywords when it began, and has only a few more now. The language works on a doubly linked ring of 64 cells and a cell pointer. Each cell can contain an unsigned byte value, that is, an integer between 0 and 255, and just as an unsigned byte would do, if the value reaches either limit, it loops around the other end. Every cell starts with an initial value of 0.

3 Keywords and Syntax

As of now there are 10 instructions in LawaUnpa, white-space in the source file is only necessary as separation and any extra white-space is ignored:

`sinpin`

Moves the cell pointer to the cell ahead of the current one.

`monsi`

Moves the cell pointer to the cell behind the current one.

`ala`

Resets the current cell value to 0.

`wan`

Adds 1 to the current cell value.

`to`

Adds 2 to the current cell value.

`luka`

Adds 5 to the current cell value.

`ike`

Sets the current cell value to it's *opposite*, that is 256 - the current value.

`unpa`

Sums the value of the cell behind the current one and the current one and places the result on the current cell.

`sike`

Marks the beginning of a loop. Checks if the current cell value is 0, if it is, it moves to the end of the loop. If it isn't, the program continues its execution normally.

`pini`

Marks the end of the current context which might refer to the last loop that was started or, if all loops have been closed, the end of the program.

If ending a loop, the instruction pointer must move to the begining of the loop.

If ending the program, it must use the current cell value as the exit code.

`toki`

Prints the Unicode character with value equivalent to the current cell's. For example, if the current cell is`65`, the program will print `A`

4 Examples

4.1 The first LawaUnpa program.

The following program outputs `alawa_to_wan_to_to`

```
luka luka wan sinpin
luka luka luka luka luka luka luka luka luka luka
luka luka luka luka luka luka luka luka luka to toki
unpa toki
monsi ike sinpin unpa toki
monsi ike luka to sinpin unpa toki
monsi ike sinpin unpa toki
monsi ala to ike sinpin unpa toki
monsi ala luka luka luka luka wan sinpin unpa toki
monsi ala luka ike sinpin unpa toki
monsi ala luka luka luka wan ike sinpin unpa toki
monsi ala luka luka luka luka to to sinpin unpa toki
monsi ala luka luka luka luka to ike sinpin unpa toki
luka luka wan to toki
monsi ala luka luka luka ike sinpin unpa toki
monsi ala luka luka luka luka wan sinpin unpa toki
monsi ala luka ike sinpin unpa toki
monsi ala luka luka luka wan ike sinpin unpa toki
monsi ala luka luka luka luka wan sinpin unpa toki
monsi ala luka ike sinpin unpa toki
sinpin pini
```

4.2 The first LawaUnpa program.

The following program outputs `ABCDEFGHIJKLMNOPQRSTIUVWXYZ`

```
luka luka to wan ike
sike
sinpin luka monsi wan
pini
sinpin sinpin ala
luka luka luka luka luka wan ike
sike
wan monsi toki wan sinpin
pini
sinpin pini
```