Luna Programming Language

Gabriel Margarido

October 6, 2022

1 Building LunaC and LPM from sources

Guessing operating system (recommended)

```
cd src && make
```

Windows for x86-64

```
cd src && make windows
```

Windows for ARM64

```
cd src && make windows-arm64
```

Linux for x86-64

```
cd src && make linux
```

Linux for ARM64

```
cd src && make linux-arm64
```

2 Using interpreted built-in LunaC - Slower

```
./lunac.sh -c <source_file>
./lpm.sh -i http://yoursite.com.br/file.ts
```

3 Using compiled LunaC - Faster

```
lunac -c <source_file>
lpm -i http://yoursite.com.br/file.ts
```

4 Code Comments

```
# This is my single line comment

# This is another comment
#This is an invalid comment
```

5 Basic Program in Luna

```
namespace "HelloWorldProgram"
        puts "Hello world"

end_namespace
```

6 Global/Public Variables

```
var [identificator] := [value]
var msg := "Hello world"
var age := 18
```

7 Local/Private Variables

```
let [identificator] := [value]
let msg := "Hello world"
let age := 18
```

8 Std I/O - Line Feed

```
puts [value/string]
puts "Hello world"
puts msg
```

9 Std Output - Std I/O

```
print [value/string]
print "Hello world"
print msg
```

10 Std Input - Std I/O

```
scanf : public [variable]
scanf : private [variable]
scanf : default [variable]
```

```
scanf : public myValue
scanf : private myValue
scanf : default myValue

puts myValue
```

11 Include directive

include ”file.luac” inside the bytecode without imports

```
include "file.luac"
include "file.lua"
```

12 Import local libraries written in Lua

```
import "[module]" as [module_nickname]
import "os" as System
```

13 Import global modules written in Lua and Luna

```
import "[module_identificator]"
import "Math"
```

14 Create global modules in Luna

```
module "[module_identificator]"
module "Math"
```

15 Declare, initialize and do references to module variables

```
var Math::MyValue := 12
let Math::MyValue := 12.67456
```

16 Declare, initialize and do references to module variables and functions

```
public def Math::Calculate(num)
       return num+5
end

call Math::Calculate(45) : result
puts result
```

17 For loop

```
for i in 1..10
     ...
end
```

18 While loop

```
while a > b
    ...
    break
end
```

19 Infinity loop

```
loop
     ...
     break
end
```

20 If-Elseif-Else Structure

```
var age := 15
if age == 18
     ...

elseif age < 5
     ...

else
     ...

end
```

21 Tables

All tables in Lua and Luna begin with index 1, instead of 0

```
let text := "Hello world"
let client := "Joseph"
public table MyTable := {text,12,35.4,client}


private table MyTable := {2,4,6,8,10}
default table MyTable := {2.8,4.7,6.2,8.9,10.3}


var myValue := MyTable[1]      # FIRST INDEX

var myValue := MyTable[0]      # ERROR
```

22 Inspect Tables

1. The first and short way of inspecting a table.

```
inspect MyTable
```

2. The second of inspecting a table and return the result to a variable.

```
inspect MyTable => InspectedTable
puts InspectedTable
```

3. Inspecting tables via function-calls

```
call inspect(MyTable) : InspectedTable
puts InspectedTable
```

23 OOP - Object Oriented Programming

```
default class [identificator]
default def [identificator].[method_identificator] ([param1],[param2],[param3], ...)
   ...
end
[identificator].[property] := [value]
[identificator].[property] := [value]
```

```
default class Dog
default def Dog.speak(phrase)
   ...
end
Dog.weight := 45
Dog.surname := "Diensberg"
```

```
public class [identificator]
public def [identificator].[method_identificator] ([param1],[param2],[param3], ...)
         ...
end
[identificator].[property] := [value]
```

```
public class Dog
public def Dog.speak(phrase)
        ...
end

Dog.weight := 45
Dog.surname := "Diensberg"
```

24 Inheritance

```
public class [class_name] extends [another_class]

private class [class_name] extends [another_class]


public class Bus extends Car
private class Bus extends Car


private class Cat extends Dog

@Override
private def Cat.speak(phrase)
      ...
end

default def Cat.meow()
       ...
end
```

25 Modificators

They give global/public access to instantiated object.

```
default new [object_identificator] : [class_identificator]
default new Tom : Dog

private new [object_identificator] : [class_identificator]
private new Tom : Dog
```

It gives local/private access to instantiated object.

```
public new [object_identificator] : [class_identificator]
public new Tom : Dog
```

26 Static properties

```
[class_identificator].[static_property] := [value]
Dog.weight := 45
Dog.name := "Tom"
```

27 Objects from classes

1. Creating new object ”jake” from class ”Dog”
   2. Here the keyword none means, that function/method does not return anything.

```
public new jake : Dog
call jake.speak() : none
```

28 Calling functions from class object

```
call [object_identificator].[method_identificator]
([param1],[param2],[param3], ...) : none

call Dog.bark() : none
call Dog.bark(a,b,c) : none
```

29 Calling function from module

```
call [method_identificator].[method_identificator]
([param1],[param2],[param3], ...) : none

call Math.calculate() : none
call Math.calculate(a,b,c) : none
```

30 Calling function with return

```
call [object_identificator].[method_identificator]
([param1],[param2],[param3], ...) : [return_identificator]

call Dog.bark() : spoke_dog
call Dog.bark(a,b,c) : spoke_dog
```

31 Calling function with return from module

```
call [method_identificator].[method_identificator]
([param1],[param2],[param3], ...) : [return_identificator]

call Math.calculate() : result
call Math.calculate(a,b,c) : result

puts result
```

32 Import modules from package manager

Download and import modules from internet

```
$> lpm -i https://yoursite.org.br/lib.lua
```

Add this line inside source-code file, corresponding to the module file name.

```
webimport "lib.lua"
```

33 Package.json on Luna (deps.config)

Install all modules in cache from ”deps.config” - Similar to Javascript’s ”package.json”

```
$> lpm --config
```

34 Uninstall packages from Luna’s local cache

```
$> lpm -u lib.luac
```

35 Macros for detecting operating system

```
@if_unix
    ...
end

@if_win32
...
end
```

36 Macros for detecting processor architecture

```
@if_intel32
    ...
end

@if_intel64
    ...
end

@if_powerpc
    ...
end

@if_arm
    ...
end

@if_mips
    ...
end
```

37 Run Shell commands

```
lvm.run "ls -a"

@if_unix
    lvm.run "ls -a"
end

@if_win32
    lvm.run "dir"
end
```

38 Calculating Factorial of a number

```
var myFactorial := fat(5)
puts myFactorial
```

39 Using Lua integration

```
var pi := math.pi
puts pi

var random := math.random(0,256)
puts random
```

40 Read File

```
io.ReadFile("text.txt") => public MyFile
puts MyFile
```

41 Write File or Create new - No line feed

Create new file if does not exist and overwrite all information inside it, however does not jump to next line.

```
io.WriteFile("text.txt") => "Hello world\n"
```

42 Write File or Create new - Do line feed

Create new file if does not exist and overwrite all information inside it, and jump to next line.

```
io.WriteFileLn("text.txt") => "Hello world"
```

43 Append File - No line feed

Add information to existing file, however does not jump to next line.

```
io.AppendFile("text.txt") => "Second time\n"
```

44 Append File - Do line feed

Add information to existing file, and jump to next line.

```
io.AppendFileLn("text.txt") => "Second time"
```

45 Strings concatenation - Without function return

Strings concatenation are done with double dots (..)
And add mathematical operation is done with plus signal (+).

```
public def say(name)
    let msg := "Hello "
    puts msg..name
end

call say("Gustavo Guanabara") : none
```

46 Strings concatenation - With do function return

Strings concatenation are done with double dots (..)
And add mathematical operation is done with plus signal (+).

```
public def say(name)
    let msg := "Hello "
    return msg..name
end

call say("Gustavo Guanabara") : person
puts person
```

47 With multiple variables

Strings concatenation are done with double dots (..)
And add mathematical operation is done with plus signal (+).

```
let msg := "Hello "
let name := "Gustavo "
let friend := "my friend!"

puts msg..name..friend
```

48 Lua integrations

```
var a := math.abs(x)
var a := math.acos(x)
var a := math.asin(x)

var a := math.atan(y,x)
var a := math.atan(x)

var a := math.ceil(x)
var a := math.cos(x)
var a := math.deg(x)
var a := math.exp(x)
var a := math.floor(x)

var a := math.fmod(x,y)
var a := math.huge

var a := math.log(x)
var a := math.log(x,base)
```

*Visit this link for more information.

49 Reserved keywords and macros

```
namespace   end_namespace   if    elsif   else   end   class    extends
public   private   default
def    var    let    module    table    at    assign    :=   while    break
for    in    print    puts    import    as    webimport    include    lvm.run
@if_unix    @if_win32    @if_intel32    @if_intel64    @if_powerpc    @if_arm
@if_mips    end   call   return    new    io.ReadFile    io.WriteFile   io.AppendFile
io.WriteFileLn    io.AppendFileLn    inspect    then     scanf
```

```
Luna Programming Language Developed by Gabriel Margarido.
October 2022   -   luna.gabrielmargarido.org

Lua Programming Language Developed by Roberto Ierusalimschy,
Waldemar Celes, Luiz Henrique Figueiredo.  -  www.lua.org
```

This documentation is dedicated to:
Phd. Professor Luis Filipe Miranda de Souza Ribeiro - ***In memoriam***
Phd. Professor Elizabete Velloso de Margarido Ribeiro
Severina Silva Velloso de Margarido
Ana Cristina Baptista Miranda de Souza Ribeiro
João Paraízo - C.E.G.
Toseli Matos Paraízo (Fortran Developer and Mechanic Engineer Student) - C.E.G.
Isaías Francisco Ferreira da Silva - M.'. M.'.
Centro C.E.G. - São Gonçalo, RJ
Capítulo Perfeita União Nº 115 - Ordem DeMolay