Lua

Yixin Li
Computer Science
University of Arizona
carolineliyixin@email.arizona.edu

Jiahui Wang
Computer Science
University of Arizona
wangj2@email.arizona.edu

April 6, 2020

Abstract

The abstract of the paper

1 Introduction

2 History

Lua is a scripting language that was first developed at the Pontificla Catholic University of Rio De Janerio in Brazil. Lua was created because back in those days when computer languages were not very flexible. Many large machines needs to be operated where running a large amount of data was dependent on the input by individual programmers. This is very demanding on the programmer, because a small mistake can have a great impact on the entire program. To simplify and to improve the safety of the program, Lua was proposed. Many machine languages use a top-down structure, which is not conducive for detail modification. Lua was created to make it simpler and faster to modify the entire program.

Lua has helped many large programs with its simplicity and portability. The development of Lua has far exceeded the most optimistic expectations, very likely because of its design principle where simplicity was enforced. Lua has been continuously modified and updated to be widely used. Besides being a leading scripting languages in gaming, it has all kinds of industrial applications including robotics, image processing, literate programming, and so on.

3 Control Structures

The control structure in Lua is fairly straight forward and common as in many popular languages. There are two types of control structure: ”if-then-else” for conditional and for, while, repeat for iteration.

All the control structure ends with a key word ”end”, except that for repeat, we need no ”end” for the iteration but with a key word ”until”. One thing to notice is that Lua treats all values beside false and nil to be true.

The only conditional control structure is *if*, and there are several ways to use it:

```lua
--example 1
if a > 0 then b = 0 end

--example 2
if a > 0 then return a else return b end

--example 3
n = 5
if n > 5 then
  print("greater than 5")
elseif n == 5 then
  print ("equal to 5")
```

```
else
  print("less than 5")
end
```

Listing 1: if-then-else

The key word here are **if**,**elseif**, **then**, **else**. All key words here have fairly unambiguous meaning, except that **then**, which just indicate the sentence after it is the ”statement”, where some actions may be following if the condition before ”then” is evaluate to true.

There are two types of for loop, the *numeric for* loop and *generic for* loop, which are shown below respectively in example 1 and example 2:

```
--example 1: numeric for
                               --syntax
for i = 0, 10, 2 do            --for var=exp1,exp2,exp3 do
  print("hello world")         --   statement
end                            --end

--example 2: generic for
days = {"Sunday", "Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday"}

for i,v in pairs(days) do
  print(i,v)
end
```

Listing 2: for loop

Numeric for loop looks a lot like for loop in python. In example 1, local variable i start at 0, and end at 10, with 2 increment on i after each iteration. The third expression here is optional as the default increment value is 1. The key word **do** separate the for loop condition and the statement, and the loop end with **end**. Generic for loop can be used for even a wilder applications, especially in looping through key-value pairs using **pairs** or **ipairs** from a table, or reading from a file (using io.lines). In example 2, i,v represent index and value of a pair in the table (similar to a dictionary in python). In this case, will print out

```
0 Sunday
1 Monday
.
.
6 Saturday
```

The structure of the while loop is very similar to many languages, as shown in the example bellow.

```
i = 0
while i < 10 do
  print("hello world")
  i += 1
end
```

Listing 3: while loop

The repeat-until is very similar to the while loop. Except what while loop begin with evaluate the condition, and only proceed into the statements if the condition first evaluate to true, while repeat-until start with proceed the statement before evaluate a condition and decide if proceed again. In another words, repeat-until will iterate at least once.

```
i = 10
repeat
  print(i)
  i = i - 1
until i == 0
```

Listing 4: repeat-until

The example above will print from 10 to 1 line by line.

4 Data Types

5 Subprograms

6 Summary

7 References

References

[1] Roberto Ierusalimschy (2016) *LaTeX: Programming in Lua*, Roberto Ierusalimschy, 4th ed.

[2] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes (2007) *LaTeX: The evolution of Lua*, Proceedings of the third ACM SIGPLAN conference on History of programming languages.

[3] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes (2006) *LaTeX: Lua5.1 reference manual*.

[4] *Lua community*. https://www.reddit.com/r/lua/

[5] *Lua projects*. http://lua-users.org/wiki/LuaProjects