Expressividade em Lua

Alunos: Karran Lemos e Rodrigo Teixeira
Professor: Francisco Sant'anna

A linguagem Lua

- Foi criada em 1993 na PUC-Rio.
- Lua é uma linguagem de programação poderosa, eficiente e leve, projetada para estender aplicações.
- Influenciada pelas linguagens Scheme, Icon e Python.
- Lua é tipada dinamicamente, interpretada e possui gerenciamento de memória automático.
- Usada em programas como Adobe Photoshop Lightroom e jogos como Civilization V.

Corotinas

- Corotinas permitem que o fluxo de controle seja passado cooperativamente entre duas funções sem que nenhuma delas tenha que ser finalizada.
- Quando uma função chama uma corotina, ela pode ser executada até um certo ponto e então ceder o controle de volta para a função que a chamou, podendo passar algum valor de volta para ela.

Corotinas

```lua
co = coroutine.create(function (inicio, fim, passo)
    for i = inicio, fim, passo do
        coroutine.yield(i*23 + 9)
        coroutine.yield(i*13 - 2)
        coroutine.yield(i*11)
    end
end)

while true do
    status, ret = coroutine.resume(co, 0, 2, 0.5)
    if ret == nil then
        break
    end
    print(status, ret)
end
```

```
true    9.0
true    -2.0
true    0.0
true    20.5
true    4.5
true    5.5
true    32.0
true    11.0
true    11.0
true    43.5
true    17.5
true    16.5
true    55.0
true    24.0
true    22.0
```

Corotinas

```lua
co = coroutine.create(function (inicio, fim, passo)
    for i = inicio, fim, passo do
        coroutine.yield(i*23 + 9)
        coroutine.yield(i*13 - 2)
        coroutine.yield(i*11)
    end
end)
```

Lua

```c
double i = 0;
int qualYield = 0;
bool corotinaComecou = false;
bool corotinaAcabou = false;

bool minhaCorotina(double inicio, double fim, double passo, bool *status, double *ret) {
    if (corotinaAcabou) {
        *status = false;
        return false;
    }
    if (!corotinaComecou) {
        i = inicio;
        corotinaComecou = true;
    }
    while (i <= fim) {
        switch (qualYield) {
            case 0:
                *ret = i*23 + 9;
                break;
            case 1:
                *ret = i*13 - 2;
                break;
            case 2:
                *ret = i*11;
                break;
            default:
                return false;
        }
        *status = true;
        qualYield = (qualYield+1) % 3;
        if (qualYield == 0)
            i += passo;
        return true;
    }

    corotinaAcabou = true;
    *status = true;
    return false;
}
```

C

Tabelas

- Estruturas de dados básica de Lua.
- Cada elemento se constitui de uma chave e um valor, que podem ser de qualquer tipo de Lua.
- Exemplos : Arrays , Objetos.

Arrays

```lua
x = { 1,2,3,7,9,10 }
y = { 2,4,9,22,15 }

print(x[1])
print(y[2])
print("Tamanho de X:" .. #x)
print("Tamanho de Y:" .. #y)
print("Soma do 3 Elemento de X + 4 Elemento de Y = "
    .. x[3]+y[4])
```

```
1
4
Tamanho de X:6
Tamanho de Y:5
Soma do 3 Elemento de X + 4 Elemento de Y = 25

```

Objetos

- Tem estado.
- Pode ter alterações de valores, mas é sempre o mesmo objeto.
- Duas tabelas com o mesmo valores são objetos diferentes.

```
23
table: 003ab828
24
table: 003ab828
Objetos diferentes
table: 003ab8c8
```

```lua
usuario = { nome = "Rodrigo", idade = 23 }
print(usuario.idade)
print(usuario)
usuario.idade = 24
print(usuario.idade)
print(usuario)

usuario_novo = { nome = "Karran", idade = 23 }

if (assert(usuario ~= usuario_novo)) then
    print("Objetos diferentes")
end

print(usuario_novo)
```

Tabelas

```lua
numero={}
table.insert(numero,10)
table.insert(numero,20)
table.insert(numero,30)
table.insert(numero,40)
table.insert(numero,50)
table.insert(numero,60)
for key in pairs(numero) do
    print(numero[key])
end
print("-----------------------------------")
print("Inserindo novo elemento na posição 6")
table.insert(numero, 6, 55)
for key in pairs(numero) do
     print(numero[key])
end
print("-----------------------------------")
print("Removendo novo elemento na posição 3")
table.remove(numero, 3)
for key in pairs(numero) do
     print(numero[key])
end
```

```
10
20
30
40
50
60
-----------------------------------
Inserindo novo elemento na posição 6
10
20
30
40
50
55
60
-----------------------------------
Removendo novo elemento na posição 3
10
20
40
50
55
60
```

Em C

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
  int *vetor;
  int i, num_componentes;
  printf("Informe o numero de componentes do vetor\n");
  scanf("%d", &num_componentes);
  vetor = (int *) malloc(num_componentes * sizeof(int));
  for (i = 0; i < num_componentes; i++){
      printf("\nDigite o valor para a posicao %d do vetor: ", i+1);
      scanf("%d",&vetor[i]);
      }
  printf("\n********** Valores do vetor dinamico ***********\n\n");
  for (i = 0;i < num_componentes; i++){
        printf("%d\n",vetor[i]);
  }
  free(vetor);
}
```

```
Informe o numero de componentes do vetor
5

Digite o valor para a posicao 1 do vetor: 1

Digite o valor para a posicao 2 do vetor: 2

Digite o valor para a posicao 3 do vetor: 3

Digite o valor para a posicao 4 do vetor: 4

Digite o valor para a posicao 5 do vetor: 5

********** Valores do vetor dinamico ***********

1
2
3
4
5
```

Porém em C, se eu quisesse aumentar, adicionando no fim da lista ou no meio um novo elemento, teríamos que fazer uma cópia do array acima para depois alocar mais memória para o novo array (elemento+array atual). Tendo assim, que reestruturar violentamente o código em LUA.