Compiladores II

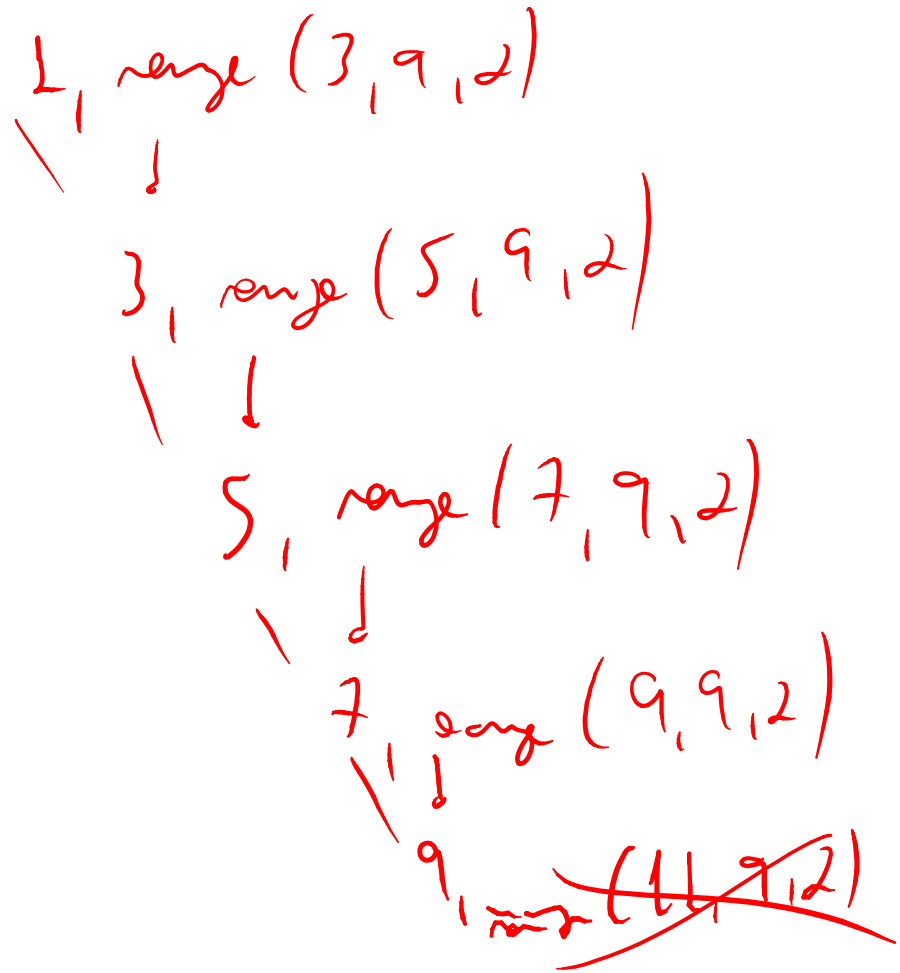
Fabio Mascarenhas - 2014.2

http://www.dcc.ufrj.br/~fabiom/comp2

Quiz

- Qual a saída do programa abaixo?

```
local function range(a, b, c)
  if a > b then
    return
  else
    return a, range(a + c, b, c)
  end
end
print(range(1, 9, 2))
```



Tabelas para tudo

- Tabelas são o único tipo estruturado de Lua

- Elas podem representar vetores, conjuntos, registros, objetos, e outras estruturas de dados de maneira eficiente, e com suporte sintático

- As operações básicas são *construção* ({}), para criar uma nova tabela, e *indexação* ([ ]), para ler e escrever valores

```
> tab = {}            -- cria nova tabela, associa a tab
> tab["x"] = 5        -- escreve 5 no campo “x”
> print(tab["x"])     -- lê valor do campo “x” e imprime
5
```

- Tabelas são um *tipo mutável por referência*, então têm os mesmos problemas de aliasing de objetos Java e pointeiros C

```
> alias = tab
> alias["x"] = "mudou"
> print(tab["x"])
mudou
```

Vetores

- Um vetor Lua é uma tabela com valores associados a chaves inteiras sequenciais, começando em 1

```
local a = {}
for i = 1, 6 do
  a[i] = math.random(10)
end
```

- Podemos inicializar um vetor usando um constructor com uma lista de expressões

```
-- um vetor como o anterior
local a = { math.random(10), math.random(10), math.random(10),
            math.random(10), math.random(10), math.random(10) }
```

- Um vetor não pode ter *buracos*: nenhum valor pode ser `nil`, mas você pode preencher o valor em qualquer ordem, contanto que todos os buracos sejam preenchidos antes de começar a usar o vetor

Tamanho de vetores

- O operador de tamanho (#) dá o número de elementos em um vetor

- Podemos usar ele para iterar sobre um vetor:

```
local a = { math.random(10), math.random(10), math.random(10),
            math.random(10), math.random(10), math.random(10) }
for i = 1, #a do
  print(a[i])
end
```

- Ele também é útil para adicionar elementos no final de um vetor, e para remover o ultimo elemento:

```
a[#a] = nil                 -- remove o ultimo elemento
a[#a + 1] = math.random(10) -- adiciona element ao final
```

Inserindo, removendo, ordenando

- Duas funções no módulo `table` inserem e removem elementos em qualquer posição de um vetor, empurrando elementos para abrir espaço, ou puxando elementos para fechar o buraco:

```
> a = { 1, 2, 3, 4, 5 }
> table.insert(a, 3, 10) -- insere 10 na posição 3
> print_array(a)
{ 1, 2, 10, 3, 4, 5 }
> table.remove(a, 4)     -- remove quarto element
> print_array(a)
{ 1, 2, 10, 4, 5 }
```

- A função `table.sort` ordena um vetor:

```
> a = { "Python", "Lua", "C", "JavaScript", "Java", "Lisp" }
> table.sort(a)
> print_array(a)
{ C, Java, JavaScript, Lisp, Lua, Python }
```

Concatenação

- A função table.concat concatena um vetor de strings usando um separador opcional:

```
function print_array(a)
  print("{" .. table.concat(a, ", ") .. "}")
end
```

- Na falta de um separador `concat` usa `""`

- Um idioma em Lua é usar um vetor de strings como um buffer, e usar table.concat quando queremos o conteúdo do buffer como uma única string

Iteração com `ipairs`

- Outro jeito de iterar sobre um vetor é usando o laço `for` *genérico* e a função embutida `ipairs`:

```
local a = { 1, 3, 5, 7, 9 }
local sum = 0
for i, x in ipairs(a) do
  print("adding element " .. i)
  sum = sum + x
end
print("the sum is " .. sum)
```

- Esse laço tem duas variáveis de controle: a primeira recebe os índices, a segunda os elementos

- É comum usar _ como a variável de controle dos índices, quando estamos interessados apenas nos elementos

Matrizes

- Uma maneira de representar vetores multi-dimensionais é com “jagged arrays”, como em Java, onde temos vetores de vetores para duas dimensões, vetores de vetores de vetores para três, etc.

```
local mt = {}
for i = 1, 3 do
  mt[i] = {}
  for j = 1, 5 do
    mt[i][j] = 0
  end
end
```

- Uma maneira mais eficiente é compor os índices com multiplicação, como os vetores de C:

```
local mt = {}
for i = 1, 3 do
  for j = 1, 5 do
    mt[(i-1)*5+j] = 0
  end
end
```

Registros

- Um registro (ou struct) Lua é uma tabela com chaves string, onde as chaves são identificadores válidos; podemos criar um registro passando pares chaves/valor no construtor:

```
point1 = { x = 10, y = 20 }
point2 = { x = 50, y = 5  }
line   = { from = point1, to = point2, color = "blue" }
```

- Podemos usar o operador `.` para acessar campos em um registro:

```
line.color = "red"                  -- same as line["color"] = "red"
print(line.from.x, line["color"])
```

- Uma tabela pode ser ao mesmo tempo um registro e um vetor, podendo misturar as duas formas de inicialização no construtor

Conjuntos

- Um jeito elegante de representar conjuntos em Lua é com uma tabela onde as chaves são os elementos do conjunto, e os valores são `true`

- O teste de pertinência no conjunto vira uma indexação, e também podemos acrescentar e remover elementos indexando a tabela

- Uma terceira sintaxe para o construtor é útil para inicializar conjuntos:

```lua
-- conjunto de quarto números inteiros aleatórios de 1 a 10
local set = { [math.random(10)] = true, [math.random(10)] = true,
              [math.random(10)] = true, [math.random(10)] = true }
```

- Substituindo `true` por um número temos um *multiconjunto*, onde cada element pode aparecer mais de uma vez

Iterando com `pairs`

- Um laço `for` usando a função embutida `pairs` itera sobre todas as chaves e valores de uma tabela:

```
local tab = { x = 5, y = 3, 10, "foo" }
for k, v in pairs(tab) do
  print(tostring(k) .. " = " .. tostring(v))
end
```

- Como em `ipairs`, a primeira variável de controle recebe as chaves, a segunda os valores

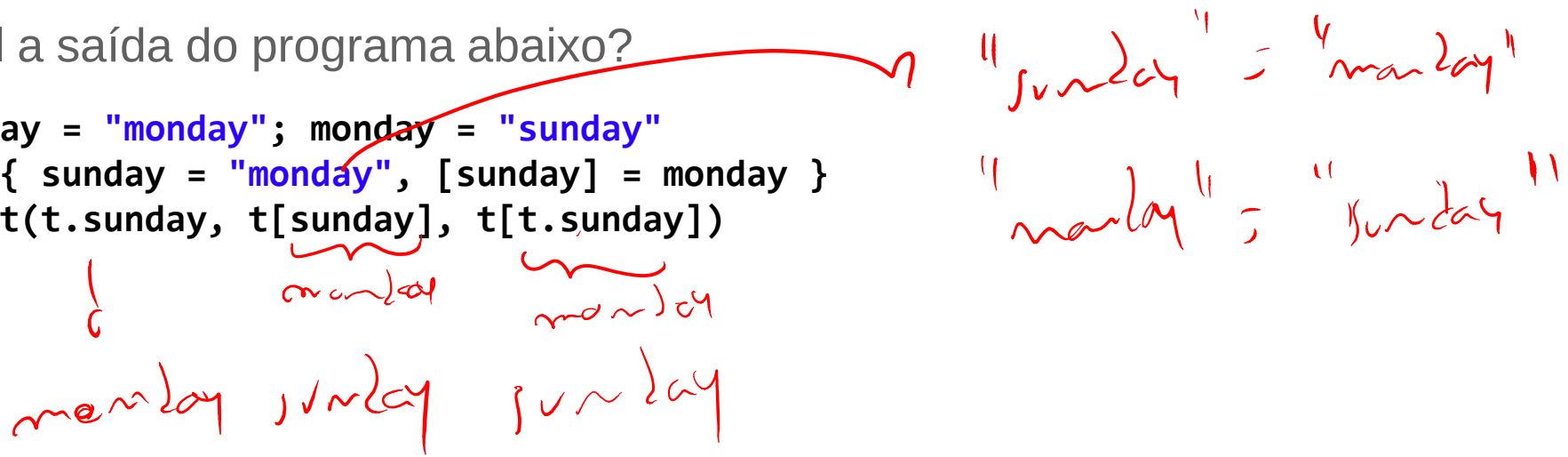
- Tabelas são tabelas hash, então `pairs` não garante a ordem de iteração; use sempre `ipairs` para iterar sobre um vetor

- Mas `pairs` é ótimo para iterar sobre um conjunto, usando _ como variável de controle para os valores

Quiz

- Qual a saída do programa abaixo?

```
sunday = "monday"; monday = "sunday"
t = { sunday = "monday", [sunday] = monday }
print(t.sunday, t[sunday], t[t.sunday])
```



Dojo

- O objetivo é construir um tokenizador para um subconjunto de Lua, a função token deve receber uma string, e retornar o primeiro token da string (ignorando espaços) e o que sobrou depois de consumir esse token; um token é um registro contendo o tipo do token e o lexema
- Palavras reservadas: apenas function, end, while, local, true, and, false, else, if, elseif, not, nil, or, return, then, do
- Numerais: apenas inteiros e decimais sem notação científica
- Strings: apenas aspas duplas
- Operadores: apenas +, -, *, /, ==, ~=, <, =, (,), {, }, ., ,, ..
- Identificadores