

Red Programming Language

Guaracy Monteiro

9 de Outubro de 2017

RASCUNHO

Conteúdo

1	Teste de livro	5
1.1	Listagens	6
1.2	Tipagem	7
2	Constantes	11
2.1	tuplas	11
3	Exemplos	13
3.1	Foo	13
3.2	Bar	14
4	Fim	19

RASCUNHO

1

Teste de livro

1.1 Listagens

Listagem 1.2.1: Função com parâmetros dinâmicos	7
Listagem 1.2.2: Tipos aceitos pela soma (+)	8
Listagem 1.2.3: Função com parâmetros tipados	8
Listagem 3.2.1: Bloco original gerado pelo org-mode	15
Listagem 3.2.2: Bloco desejado para o .tex	15
Listagem 3.2.3: Script para alteração do .tex	16

Última versão em: <https://guaracy.github.io/docs/gbm.pdf>

1.2 Tipagem

Apesar de não fazer distinção entre códigos e dados (homoiconicidade) os dados possuem tipos. Na realidade, existe uma grande variedade de tipos para facilitar a programação e a legibilidade. Para referenciar um determinado valor, não é necessário especificar o tipo do mesmo. Mesmo na declaração de uma função, a declaração dos tipos é opcional. Se a operação entre os valores não for permitida, o programa irá parar informando um erro. Por exemplo:

Listagem 1.2.1: Função com parâmetros dinâmicos

```
1 soma: func[a b][
2     a + b
3 ]
4 print soma 1 2
5 print soma 1.5 2.6
6 print soma 2x4 4x4
7 data: now
8 print data
9 print soma data 26:10:10
10 print soma "no" "me"

3
4.1
6x8
9-Oct-2017/12:05:56-03:00
10-Oct-2017/14:16:06-03:00
*** Script Error: + does not allow string! for its value1 argument
*** Where: +
*** Stack: soma
```

Como podemos ver, a chamada ocorreu normalmente mas, no momento em que passamos duas strings como parâmetros, o programa mostrou um erro dizendo que a operação soma não aceitava tipos string. Vejamos os tipos aceitados por `+`:

Listagem 1.2.2: Tipos aceitos pela soma (+)

```
help +  
-----  
USAGE:  
    value1 + value2  
  
DESCRIPTION:  
    Returns the sum of the two values.  
    + is an op! value.  
  
ARGUMENTS:  
    value1      [number! char! pair! tuple! vector! time! date!]  
    value2      [number! char! pair! tuple! vector! time! date!]  
  
RETURNS:  
    [number! char! pair! tuple! vector! time! date!]
```

Podemos ver que a soma não aceita parâmetros do tipo **string!**. Da mesma forma, podemos especificar os tipos dos parâmetro aceitos por uma função que definimos. Por exemplo, se desejamos que aceite apenas valores inteiros, percentuais e hora (ordens superiores), declaramos como segue:

Listagem 1.2.3: Função com parâmetros tipados

```
1 soma: func[a [integer! percent! time!] b [integer! percent! time!]] [  
2     a + b  
3 ]  
4 print soma 1 2  
5 print soma 10:05:22 0:5:40  
6 print soma 2x3 6x7  
  
-----  
3  
10:11:02  
*** Script Error: soma does not allow pair! for its a argument  
*** Where: soma  
*** Stack: soma
```

Quando ela foi chamada com um par, não chegou a executar a soma,

acusando o erro no momento da chamada da função.

RASCUNHO





RASCUNHO

2

Constantes

2.1 tuplas

Tuplas definidas na linguagem:

Nome	Tupla	Cor	Nome	Tupla	Cor
red	255.0.0		aqua	40.100.130	
beige	255.228.196		black	0.0.0	
blue	0.0.255		brick	178.34.34	
brown	139.69.19		coal	64.64.64	
coffee	76.26.0		crimson	220.20.60	
cyan	0.255.255		forest	0.48.0	
gold	255.205.40		gray	128.128.128	
green	0.255.0		ivory	255.255.240	
khaki	179.179.126		leaf	0.128.0	
linen	250.240.230		magenta	255.0.255	
maroon	128.0.0		mint	100.136.116	
navy	0.0.128		oldrab	72.72.16	
olive	128.128.0		orange	255.150.10	
papaya	255.80.37		pewter	170.170.170	
pink	255.164.200		purple	128.0.128	
reblue	38.58.108		rebolor	142.128.110	
sienna	160.82.45		silver	192.192.192	
sky	164.200.255		snow	240.240.240	
tanned	222.184.135		teal	0.128.128	
violet	72.0.90		water	80.108.142	
wheat	245.222.129		white	255.255.255	
yello	255.240.120		yellow	255.255.0	
glass ¹	0.0.0.255		transparent	0.0.0.255	

¹glass e transparent definem uma tupla no formato **r.g.b.a** onde **a** é o nível de transparência da cor.

3

Exemplos

3.1 Foo

Exemplo tirado do [site](#) de **Red** . Apesar dos tons vermelhos nos outros quadros com referência ao nome da linguagem, este está em preto e branco para testar.

Listagem : Exemplo em GTK3

```
langs: ["English" "French" "Português"]
labels: [
["Name" "Age" "Phone #" "Cancel" "Submit"]
["Nom" "Age" "Tél." "Abandon" "Envoyer"]
["Nome" "Idade" "Telefone" "Cancelar" "Enviar"]
]
set-lang: function [f event] [
root: f/parent
condition: [all [face/text face/type <> 'drop-list]]

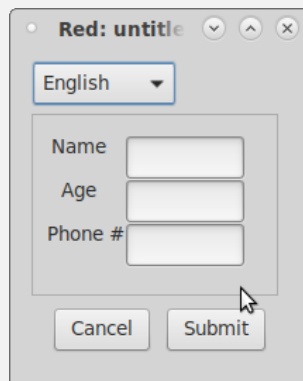
list: collect [foreach-face/with root [keep face/text] condition]
forall list [append clear list/1 labels/(f/selected)/(index? list)]

foreach-face/with root [
pads: any [metrics?/total face 'paddings 'x 0]
prev: face/size/x
```

```

face/size/x: pads + first size-text face
face/offset/x: face/offset/x + ((prev - face/size/x) / 2)
][face/type = 'button]
]
view [
style txt: text right 45
drop-list data langs select 1 on-change :set-lang return
group-box [
txt "Name" field return
txt "Age" field return
txt "Phone" field
] return
pad 15x0 button "Cancel" button "Submit"
]

```



3.2 Bar

Pela facilidade de trabalhar com org-mode, toda documentação foi desenvolvida com ele. A geração do pdf também poderia ter sido mas, não gostei exatamente do resultado da dobradinha código/resultado. Resolvi utilizar o tcolorbox para a apresentação da listagem dos fontes e apresentação dos resultados. Desenvovi normalmente e gerava o pdf para visualização com a apresentação padrão. O fonte gerado pelo org-mode para o latex era assim:

Listagem 3.2.1: Bloco original gerado pelo org-mode

```
\begin{listing}[htbp]
\begin{minted}[linenos,firstnumber=1]{red}
soma: func[a b][
    a + b
]
print soma 1 2
print soma 1.5 2.6
print soma 2x4 4x4
print soma "no" "me"
\end{minted}
\caption{Função com parâmetros dinâmicos}
\end{listing}
```

Eu precisava de um formato compatível com o `tclobox` e com as seguintes características. Chamaria uma macro `codeFromFile` com os seguintes parâmetros:

- linguagem para salientar a sintaxe
- nome do arquivo fonte para ser incluído (estaria da pasta `listings`)
- título para a listagem
- referência para entrar no índice
- parâmetros adicionais para o `minted` gerados pelo `org-mode`

No final teria algo assim:

Listagem 3.2.2: Bloco desejado para o .tex

```
\codeFromFile{red}{listings/p3.red}{Função com parâmetros
↪ tipados}{code3}{linenos,firstnumber=1}
```

O script (em **Red** é claro) para processar o arquivo `.tex` e efetuar as alterações ficou assim (bacalhau desenvolvido em minutos):

Listagem 3.2.3: Script para alteração do .tex

```

1 doc: read %teste.tex
2
3 z: copy ""
4 ip: 0
5
6 f: func[s][
7     ip: ip + 1
8     print rejoin ["gerando: programa" ip "..."]
9     parse s [
10         thru "minted{" copy mint-style to "]"
11         skip copy prg-ext thru "}"
12         thru lf copy prg-src to "\end{minted}"
13         thru "\caption{" copy rem to "}"
14     ]
15     pgs: rejoin ["listings/p" ip ".red"]
16     write to file! pgs prg-src
17     either rem/1 = #"*" [
18         remove rem
19         cmd: "\codeFromFileOnly"
20     ] [
21         cmd: "\codeFromFile"
22         insert prg-src "Red []~/\"
23         write %xyz.red prg-src
24         out: copy ""
25         call/shell/console/output "red xyz.red" out
26         pgr: rejoin ["listings/p" ip ".res"]
27         write to file! pgr out
28     ]
29     z: rejoin [cmd prg-ext "{" pgs "}" rem "{code" ip "}"
30     ↪ mint-style "}"]
31 ]
32 rule: [
33     any [
34         to "\begin{listing}" p: insert (f p) remove thru
35         ↪ "\end{listing}"
36     ]
37 ]
38 parse doc rule
39 write %gbm.tex doc
40 print "Gerando pdf..."
41 call/shell/console "pdflatex -shell-escape gbm.tex"

```


- **doc** : leitura do arquivo de teste (hardcoded por enquanto)
- **z** : conterá o exto que substituirá o bloco do arquivo original
- **ip** : índice utilizado na geração do nome do arquivo que conterá o fonte e o rótulo para o índice das listagens
- **f** : função que recebe um ponteiro para o início do casamento da ocorrência analisando a entrada e:
 - armazenando estilo para minted em **mint-style**
 - armazenando tipo do programa em **prg-ext**
 - armazenando fonte do programa em **prg-src**
 - armazena rótulo da caixa em **rem**
 - se rótulo inicia com asterisco, não gera a saída do programa (solução encontrada no momento)
 - gera arquivo **xyz.red**, executa, captura a saída e gera arquivo **.red** com o resultado
 - gera texto para substituir a macro no arquivo **.tex**
- **rule** : se texto casa, chama **f** com ponteiro para o início do bloco, insere nova macro e elimina texto até o final do bloco
- grava em **gbm.tex** novo conteúdo
- executa **pdflatex** para gerar o pdf.

Melhorias para próxima versão (bacalhau 2.0):

- se md5 do fonte for diferente/não existe então proceder na gravação do fonte e saída, caso contrário eliminar estas duas etapas.
- programas gráficos, executar `/no-wait` e salvar screenshot com o nome do programa.png (criar macro adequada para o latex)

RASCUNHO

4

Fim

Terminou por enquanto. Algumas coisas ainda precisam de ajustes como o índice das listagens gerado pelo tcolorbox. Outros problemas poderão ser encontrados. Mas é uma opção interessante entrar o documento diretamente em uma linguagem de marcação mais simples que o \LaTeX , com uma visualização melhor e gerar um pdf de qualidade aceitável.