MAC110 - Introdução à Computação

Anderson Andrei da Silva (silva.andersonandrei@gmail.com),
Professor Alfredo Goldman (gold@ime.usp.br)

24 de Junho de 2020

Aula de hoje:

Strings:

- O que são *strings*?
- O tipo *char*,
- Iteração,
- Partições ou fatias,
- Concatenação e interpolação,
- Laços e contadores,
- A biblioteca String,
- O operador $\in$,
- Comparação de strings,
- Imutabilidade,
- Exercícios.

Livro ThinkJulia (PenseJulia)

- Essa aula é uma adapatação do capítulo 8, Strings, do livro ThinkJulia, disponível em: `https://benlauwens.github.io/ThinkJulia.jl/latest/book.html`
- Onde existe uma versão, em produção, traduzida e interativa via Jupyter Notebook: `https://phrb.github.io/PenseJulia/`

Strings

Strings são sequências de caracteres. Ou seja, um conjunto ordenado de valores. Esses valores, por sua vez, caracteres, são um tipo específico de dados, o tipo Char.

Caracteres

Entendemos por caracteres:

- Letras do alfabeto (A, B, ..., Z),
- Numerais,
- Algumas pontuações.

Uma das formas de computar tais caracteres é através do padrão ASCII (American Standard Code for Information Interchange):

- Esse padrão mapeia os caracteres dentre valores de 0 à 127. Dentro dessa sequência temos, por exemplo:
  - O alfabeto, em maiúsculo, começando de A(65) à Z(90),
  - O alfabeto, em minúsculo, começando de a(97) à z(122)

Para mais alguns detalhes, consultem: `https://pt.wikipedia.org/wiki/ASCII`

Caracteres

Uma das formas de computar tais caracteres é através do padrão ASCII (American Standard Code for Information Interchange):

- Esse padrão mapeia os caracteres dentre valores de 0 à 127. Dentro dessa sequência temos, por exemplo:
  - O alfabeto, em maiúsculo, começando de A(65) à Z(90),
  - O alfabeto, em minúsculo, começando de a(97) à z(122)

```
julia> 'x'
'x': ASCII/Unicode U+0078 (category Ll: Letter, lowercase)

julia> 'X'
'X': ASCII/Unicode U+0058 (category Lu: Letter, uppercase)

julia> typeof('x')
Char
```

String: Uma sequência de caracteres

Uma string é uma sequência de caracteres, ou seja um conjunto ordenado de caracteres. Sendo assim, podemos acessar tais valores da mesma forma que acessamos uma outra sequência de valores ao qual já estamos acosutmados, vetores:

```
julia> fruit = "banana"
"banana"

julia> letter = fruit[1]
'b': ASCII/Unicode U+0062 (category Ll: Letter, lowercase)

julia> fruit[end]
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase
```

Note que: Trabalhamos em Julia com indexação começando por 1, e terminando com end.

String: Uma sequência de caracteres

E já que acessamos os caracteres de uma string via indexação, podemos usar variáveis para manipular tais itens, como por exemplo:

```
julia> i = 1
1

julia> fruit[i+1]
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)

julia> fruit[end-1]
'n': ASCII/Unicode U+006e (category Ll: Letter, lowercase)
```

Note que: Tais variáveis para indexação devem ser do tipo inteiro.

A função *length*

A função *length* retorna o número de caracteres em um string. Por exemplo:

```
julia> fruta = "maçã"
julia> length(fruta)
4
```

Mas, cuidado, pois ao tentarmos usar tal valor para acessar os caracteres dessa string, podemos nos deparar com o seguinte erro:

```
julia> fruta[4]
ERROR: StringIndexError("maçã", 4)
```

A função *sizeof*

Strings são codificadas a partir do padrão UTF-8, que codifica caracteres com tamanhos variados. Além disso, Julia indexa as strings a partir do tamanho (bytes) de cada caractere no padrão UTF-8.

A função *sizeof* nos retorna o tamanho, em bytes, de um caractere, vejamos:

```
julia> sizeof("m")
1


julia> sizeof("ç")
2


julia> sizeof("maçã")
6
```

Ou seja, o índice para acessar o caractere "ç" é o 3, mas para acessar o "ã" é o 5, pois devemos considerar o índice do "ç" + seu tamanho em bytes: $3 + 2 = 5$.

A funções *nextind*

Então, para facilitar o uso de indexação com strings, existe a função *nextind*, que recebe parâmetros a e b, e retorna o próximo índice da string a, a partir do índice b. Por exemplo:

```
julia> nextind(fruta, 1)
2

julia> nextind(fruta, 3)
5
```

Iteração em strings

Muitas operações envolvem o processamento de strings, caractere à caractere, geralmente iniciando do primeiro deles.

Uma das formas de fazer isso é através de um laço utilizando a função *while*, como por exemplo:

```
index = firstindex(fruits)
while index <= sizeof(fruits)
    letter = fruits[index]
    println(letter)
    global index = nextind(fruits, index)
end
```

Exercício: O que faz o trecho de código acima?

Note que: A função *firstindex* retorna o valor do primeiro índice válido de um string.

Iteração em strings

Muitas operações envolvem o processamento de strings, caractere à caractere, geralmente iniciando do primeiro deles.

Uma das formas de fazer isso é através de um laço utilizando a função *while*, como por exemplo:

```
index = firstindex(fruits)
while index <= sizeof(fruits)
    letter = fruits[index]
    println(letter)
    global index = nextind(fruits, index)
end
```

Exercício: Escreva uma função que recebe uma string e imprime cada um de seus caracteres, de trás para frente, utilizando a função *while*.

Dica: Será que existe uma função que faz o oposto da função *nextind*?

Iteração em strings

Outra forma de laço já conhecida é através da função *for*. Sendo assim, também podemos utilizá-la para iterar em strings. Por exemplo:

```
for letter in fruits
    println(letter)
end
```

Exercício: O que faz o trecho de código acima?

Iteração em strings

Outra forma de laço já conhecida é através da função *for*. Sendo assim, também podemos utilizá-la para iterar em strings. Por exemplo:

```
for letter in fruits
    println(letter)
end
```

Exercício: Como adaptar o trecho acima para imprimir os caracteres de uma string, de trás para frente?

Concatenação de strings

Chamamos de concatenação a combinação de duas ou mais strings, de forma à colocar uma string logo no final da string anterior. Por exemplo:

```
a = 'Gostamos de '
b = 'MAC110 '
c = 'e '
d = 'Julia '
```

Assim, poderíamos fazer as seguintes concatenações:

- a e b formaria a string: 'Gostamos de MAC110 ',
- b e c formaria: 'MAC110 e Julia ',
- a e d formaria: 'Gostamos de Julia ',
- a, b, c e d: 'Gostamos de MAC110 e Julia ',
- a, b, d e c: 'Gostamos de Julia e MAC110 '.

Concatenação de strings

Chamamos de concatenação a combinação de duas ou mais strings, de forma à colocar uma string logo no final da string anterior. Por exemplo:

Em Julia, efetuamos uma concatenação através do operador/ símbolo: *. Assim sendo, nosso exemplo ficaria:

```
string a = 'Gostamos de '
string b = 'MAC110 '
string c = 'e '
string d = 'Julia'

println(a * b)
'Gostamos de MAC110'

println(a * b * c * d)
'Gostamos de MAC110 e Julia '
```

Concatenação de strings

Exercício: O que faz o seguinte trecho de código?

```
prefixes = "JKLMNOPQ"
suffix = "ack"

for letter in prefixes
    println(letter * suffix)
end
```

Concatenação de strings

Exercício: O que faz o seguinte trecho de código?

```
prefixes = "JKLMNOPQ"
suffix = "ack"

for letter in prefixes
    println(letter * suffix)
end
```

Resposta: Ele concatena a string "ack"à cada uma das letras (caracteres) da string "prefixes", produzindo: Jack Kack Lack Mack Nack Oack Pack Qack

Concatenação de strings

Exercício: Como modificar o trecho de código abaixo para que ele imprima a string 'Ouack' e 'Quack' ao invés de 'Oack' e 'Qack', respectivamente?

```
prefixes = "JKLMNOPQ"
suffix = "ack"

for letter in prefixes
    println(letter * suffix)
end
```

Interpolação de strings

No nosso exemplo de concatenação inserimos espaços ao final de cada string, para concatená-las diretamente. Caso não fizessemos, teríamos que operar da seguinte forma:

```
string a = 'Gostamos de'
string b = 'MAC110'
string c = 'e'
string d = 'Julia'

combinacao = a * ' ' * b * ' ' * c * ' ' * d
```

Interporlação é a combinação de strings, existindo a possibilidade da inserção de uma no meio de outras. Assim, utilizamos o operador $ para trazer o valor de uma string dentro de outras.

Então, o exemplo anterior poderia se escrito como:

```
combinacao = "$a $b $c $d"
```

Interpolação de strings

Interporlação é a combinação de strings, existindo a possibilidade da inserção de uma no meio de outras. Assim, utilizamos o operador $ para trazer o valor de uma string dentro de outras.

```
julia> greet = "Hello"
"Hello"

julia> whom = "World"
"World"

julia> "$greet, $(whom)!"
"Hello, World!"
```

Partição de strings

Uma partição ou fatia de uma string é um segmento, com início e fim especificado. Realizamos tal particionamento, ou "fatiamos"uma string, através da seguinte sintaxe :

nome_string[n:m], onde:

- n é o índice do primeiro byte desejado,
- m é o índice do último byte desejado.

Note que: Se o primeiro índice for maior que o segundo, o retorno será uma string vazia representada por .

Partição de strings

Uma partição ou fatia de uma string é um segmento, com início e fim especificado. Por exemplo:

```
julia> str = "Julius Caesar"

julia> str[1:6]
"Julius"

julia> str[8:end]
"Caesar"

julia> str[8:7]
""
```

Note que: Também é possível utilizar a palavra reservada end.

Partição de strings

Uma partição ou fatia de uma string é um segmento, com início e fim especificado. Por exemplo:

```
julia> str = "Julius Caesar"

julia> str[1:6]
"Julius"

julia> str[8:end]
"Caesar"

julia> str[8:7]
""
```

Exercício: Seja a string str = 'Julius Caesar', o que retornaria str[:] ?

Busca

Uma das aplicações mais comuns em strings são as buscas. Ou seja, é recorrente a necessidade de saber se um determinado caractere ou string está dentro de outra string.

Exercício: Construir uma função que recebe dois parâmetros, palavra (string) e letra (char), e busca a letra na string palavra. A função deve retornar o índice da primeira aparição de tal letra na palavra passada, caso exista ao menos uma, ou, retornar -1 caso não exista aquela letra naquela palavra.

Busca

Uma das aplicações mais comuns em strings são as buscas. Ou seja, é recorrente a necessidade de saber se um determinado caractere ou string está dentro de outra string.

Exercício: Modifique a função construída no exercício anterior. Acrescente dois parâmetros, inicio (integer) e fim (integer), representando os indíces ao qual a letra deve ser buscada na palavra passada.

A biblioteca String

Julia oferece várias funções para a manipulação de strings. Como por exemplo:

lowercase:

```
julia> lowercase("Hello, World!")
"hello, world!"
```

uppercase:

```
julia> uppercase("Hello, World!")
"HELLO, WORLD!"
```

Note que: As funções uppercase e lowercase são muito úteis para comparação de strings, já que Julia é *case sensitive*.

A biblioteca String

Julia oferece várias funções para a manipulação de strings. Como por exemplo:

findfirst:

```
julia> findfirst("a", "banana")
2:2

julia> findfirst("na", "banana")
3:4
```

findnext:

```
julia> findnext("na", "banana", 4)
5:6
```

Iteração e contadores

Para calcular a número de repetições de um caractere em uma string, podemos utilizar a estrutura de busca e incrementar uma variável a cada iteração em que encontrarmos tal caractere. Por exemplo:

```
word = "banana"
counter = 0
for letter in word
  if letter == 'a'
    global counter = counter + 1
  end
end
println(counter)
```

Note que: A variável que incrementamos a cada encontro do caractere buscado é chamada de contador.

Iteração e contadores

Para calcular a número de repetições de um caractere em uma string, podemos utilizar a estrutura de busca e incrementar uma variável a cada iteração em que encontrarmos tal caractere. Por exemplo:

```
word = "banana"
counter = 0
for letter in word
  if letter == 'a'
    global counter = counter + 1
  end
end
println(counter)
```

Exercício: Modifique o trecho de código acima e o transforme em uma função que recebe dois parâmentros, palavra (string) e letra(char), e retorna a quantidade de repetições da letra na palavra passada.

O operador ∈

O operador ∈ recebe dois parâmetros, um caractere e uma string, e retorna *true* caso esse a string contenha o caractere. Caso contrário, retorna *false*. Por exemplo:

```
julia> 'a' (\in tab) "banana"    # 'a' in "banana"
true
```

Exercício: O que faz o seguinte trecho de código?

```
function inboth(word1, word2)
  for letter in word1
    if letter (\in tab) word2
      print(letter, " ")
    end
  end
end
```

Comparação de strings

Também é muito comum compararmos strings. Para isso, podemos utilizar os operadores de comparação '>', '<', '!=' e '=', mas com os seguintes objetivos:

'=' e '!=' verificam se duas strings são iguais ou não:

```
word = "Pineapple"
if word == "banana"
    println("All right, bananas.")
end
```

Comparação de strings

Também é muito comum compararmos strings. Para isso, podemos utilizar os operadores de comparação '>', '<', '!=' e '=', mas com os seguintes objetivos:

'>' e '<' verificam qual string vem primeiro, em ordem alfabética:

```
if word < "banana"
    println("Your word, $word, comes before banana.")
elseif word > "banana"
    println("Your word, $word, comes after banana.")
else
    println("All right, bananas.")
end
```

Note que: Em Julia, letras maiúsculas tem precedência à letras as minúsuclas.

Imutabilidade

Strings são imutáveis, ou seja, não podem ser alteradas:

```
julia> greeting = "Jello, world!"
"Jello, world!"

julia> greeting[1] = 'H'
ERROR: MethodError: no method matching setindex!(::String, ::Char, ::Int64)
```

Uma alternativa, é a criação de uma nova string, utilizando a "fatia"que se deseja manter da string anterior. Por exemplo:

```
julia> greeting = "H" * greeting[2:end]
"Hello, world!"
```

Exercícios

1. Leia a documentação de Julia e teste algumas funções para manipulação de string, como por exemplo, as funções *strip* e *replace*.
2. Verifique também como funciona a função *count* e teste-a.
3. Para se obter uma partição ou fatia de uma string podemos utilizar um terceiro índice, no formato: string_name[a:b:c]. O que mudou? Faça alguns testes.

Referências

https://benlauwens.github.io/ThinkJulia.jl/latest/book.html#chap08

MAC110 - Introdução à Computação

Anderson Andrei da Silva (silva.andersonandrei@gmail.com),
Professor Alfredo Goldman (gold@ime.usp.br)

24 de Junho de 2020