I/O básico

As operações mais básicas de I/O são: ler e escrever na console. Escrever na console já sabemos, pois uso o método **fmt.Println()** desde o início. Só que ele não permite formatar strings nem números. Vamos ver outros métodos de **fmt**, como: **Printf()** e **Sprintf()**.

A leitura de dados da console consiste basicamente em **strings**. É claro que podemos especificar outros tipos de dados, pois o pacote **bufio** permite isso, mas basicamente usamos strings. Vamos ver como isso funciona. No programa exemplo lemos um nome da console:

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strings"
)

func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Your name: ")
    text, _ := reader.ReadString('\n')
    ...
```

Ele espera que você digite um nome e tecle . Seu string conterá tudo até o caracter . O método **ReadString()** retorna o string lido e um código de erro, que só acontece se o string não contiver o delimitador. Nesta caso, mandamos ler até encontrar um . Poderíamos usar qualquer caractere, como um "$":

```
text, _ := reader.ReadString('$')
```

Ele lerá tudo até encontrar um caractere "$". Note que usei aspas simples, pois é um **rune** ou um inteiro de 32 bits representando um caractere UTF-8.

E podemos mostrar o que foi digitado. Se fizermos com **fmt.Println()** sairá assim:

```
fmt.Printf("Hello, %v how are you today?\n", text)
...
Hello, Cleuton
 how are you today?
```

Aqui eu usei o método **Printf()** que permite formatarmos dados. Você pode consultar os **formatos válidos nesta página**, mas vou resumir aqui:

Formato	Descrição
%v	Qualquer valor
%t	true ou false
%d	Inteiro em base 10
%f	Números reais

Por que esta quebra de linha após o nome? A razão é que o caractere '\n', digitado por você, faz parte do string. Podemos usar a biblioteca **strings** e remover os espaços iniciais e finais do texto digitado:

```
newName := fmt.Sprintf("%v", strings.TrimSpace(text))
fmt.Printf("Now, %v, without the extra line\n", newName)
...
Now, Cleuton, without the extra line
```

E se quisermos ler números? Por exemplo, números inteiros? Simples:

```
fmt.Print("Now, type an integer: ")
nints, _ := reader.ReadString('\n')
nint, errInt := strconv.ParseInt(strings.TrimSpace(nints), 10, 64)
if errInt != nil {
    log.Fatal("Invalid integer")
}
fmt.Printf("You typed: %d\n", nint)
```

Lemos um string, eliminamos os **whitespaces** e transformamos em inteiro 64 bits de base 10 com o método **ParseInt()**. Mas temos que nos lembrar de testar se houve erro, pois o usuário pode digitar um número inválido. Lembra-se do **log.Fatal()**? Ele termina a execução com a mensagem de erro.

E números reais?

```
// Floating point
fmt.Print("Now, type a float: ")
nfs, _ := reader.ReadString('\n')
nf, errf := strconv.ParseFloat(strings.TrimSpace(nfs), 64)
if errf != nil {
    log.Fatal("Invalid float number")
}
fmt.Printf("You typed: %f\n", nf)
```

Desta forma, você deve digitar **ponto decimal** no lugar de vírgula!

i18n Go não tem suporte direto à internacionalização, mas existem pacotes que podem ajudar, como o **golang.org/x/text**. Você precisa instalá-lo com:

```
go get -v golang.org/x/text
```

Mas, para isto, precisa criar e configurar seu **GOPATH**:

1. Crie uma pasta **go** dentro da sua pasta HOME;
2. Crie uma variável de ambiente **GOPATH** e aponte para ela;
3. Modifique seu **profile** para acrescentar essa variável de ambiente.

Eis um exemplo utilizando formatação em Português do Brasil:

```
import (
    ...
    "golang.org/x/text/language"
    "golang.org/x/text/message"
)
...
p := message.NewPrinter(language.BrazilianPortuguese)
p.Printf("%f", nf)
...
10,343400
```

Arquivos de texto

Vou mostrar o I/O básico utilizando arquivos de texto. Primeiramente, vamos criar um arquivo de texto:

```
// Creating a text file
stringArr := []byte("Bom dia.\nComa uma maçã!\nTenha um ótimo dia!\n")
// Permission: -rw-r--r--
err := ioutil.WriteFile("/tmp/arq.txt", stringArr, 0644)
check(err)
...
cat /tmp/arq.txt
Bom dia.
Coma uma maçã!
Tenha um ótimo dia!
```

Como pode ver, o método **ioutil.WriteFile()** grava um array de bytes. Como usamos Unicode, ele consegue gravar os acentos sem problemas. A permissão é a mesma que usaríamos no comando **chmod**, do Linux. Eu criei uma função para verificar erros (**check**) que usa o comando **panic** em caso de problemas. Pessoalmente, prefiro o **log.Fatal()**.

Agora vamos ler este arquivo:

```
// Reading a text file
data, err := ioutil.ReadFile("/tmp/arq.txt")
```

```
check(err)
fmt.Printf("\nType: %T\n", data)
textContent := string(data)
fmt.Println(textContent)
...
Type: []uint8
Bom dia.
Coma uma maçã!
Tenha um ótimo dia!
```

Quando lemos um arquivo, o tipo de dados da variável é array de bytes (unsigned 8 bits). Podemos converter em **string** e usar normalmente.

Desafio

Crie um programa que leia da console os valores dos coeficientes de uma equação do segundo grau, calcule as raízes e exiba na tela, gravando um arquivo com cada cálculo.