Sintaxe e Semântica das LPs com Exemplo de Código Linguagem GO

Arthur Gomes[1]**,Uriel Andrade**[1]**, Vinícius Giovanini**[1]

[1]Instituto de Ciências Exatas e Informatica
Pontifícia Universidade Católica de Minas Gerais (PUCMG)
Belo Horizonte – MG – Brazil

{aosgomes,uandrade,vgiovanini}@sga.pucminas.br

1. Exemplo de Códigos em Golang

1.1. Hellow World

```
pacote  principal


importar  "fm"

func  main ()  {
    fmt . Println ( "olá mundo" )
}
```

Nosso primeiro programa imprimirá a clássica mensagem “hello world”. Aqui está o código fonte completo.

1.2. Condicionais

```
pacote  principal

importar  "fm"

func  principal ()  {

    if  7 % 2  ==  0  {
        fmt . Println ( "7 é par" )
    }  else  {
        fmt . Println ( "7 é ímpar" )
    }

    if  8 % 4  ==  0  {
        fmt . Println ( "8 é divisível por 4" )
    }

    se  num  :=  9 ;  num  <  0  {
        fmt . Println ( num ,  "é negativo" )
    }  else  if  num  <  10  {
        fmt . Println ( num ,  "tem 1 dígito" )
    }  else  {
        fmt . Println ( num ,  "tem vários dígitos" )
    }
}
```

A ramificação com if e else em Go é direta. Você pode ter uma if declaração sem um else, e aqui está um exemplo básico.

1.3. Functions

As funções são centrais em Go. Aprenderemos sobre funções com alguns exemplos diferentes. Aqui está uma função que recebe dois se int retorna sua soma como um int. Go requer retornos explícitos, ou seja, não retornará automaticamente o valor da última expressão.

```
package main

import "fmt"

func plus(a int, b int) int {


    return a + b
}

func plusPlus(a, b, c int) int {
    return a + b + c
}



func main() {

    res := plus(1, 2)
    fmt.Println("1+2 =", res)

    res = plusPlus(1, 2, 3)
    fmt.Println("1+2+3 =", res)
}
```

1.4. Arrays

Em Go, um array é uma sequência numerada de elementos de um comprimento específico. Aqui criamos um array aque irá armazenar exatamente 5 ints. O tipo de elementos e o comprimento fazem parte do tipo do array. Por padrão, um array tem valor zero, o que para ints significa 0s. Também podemos definir um valor em um índice usando a array[index] = value sintaxe e obter um valor com array[index]

```go
package main

import "fmt"

func main() {

    var a [5]int
    fmt.Println("emp:", a)



    a[4] = 100
    fmt.Println("set:", a)
    fmt.Println("get:", a[4])

    fmt.Println("len:", len(a))

    b := [5]int{1, 2, 3, 4, 5}
    fmt.Println("dcl:", b)

    var twoD [2][3]int
    for i := 0; i < 2; i++ {
        for j := 0; j < 3; j++ {
            twoD[i][j] = i + j
        }
    }
    fmt.Println("2d: ", twoD)
}
```

1.5. Loop/for

For é a única construção de loop de Go. Aqui estão alguns tipos básicos de for loops. O tipo mais básico, com uma única condição. For um loop inicial/condição/depois clássico, for sem uma condição fará um loop repetidamente até que você breaksaia do loop ou return da função delimitadora.

```go
package main

import "fmt"

func main() {

    i := 1
    for i <= 3 {
        fmt.Println(i)
        i = i + 1
    }

    for j := 7; j <= 9; j++ {
        fmt.Println(j)
    }

    for {
        fmt.Println("loop")
        break
    }

    for n := 0; n <= 5; n++ {
        if n%2 == 0 {
            continue
        }
        fmt.Println(n)
    }
}
```

1.6. Multiple Return Values

Go tem suporte integrado para vários valores de retorno . Esse recurso é usado frequentemente em Go idiomático, por exemplo, para retornar valores de resultado e de erro de uma função. Aqui usamos os 2 valores de retorno diferentes da chamada com atribuição múltipla . Se você quiser apenas um subconjunto dos valores retornados, use o identificador em branco ”Underline”.

```go
package main

import "fmt"

func vals() (int, int) {
    return 3, 7
}

func main() {

    a, b := vals()
    fmt.Println(a)
    fmt.Println(b)

    _, c := vals()
    fmt.Println(c)
}
```

1.7. Pointers

Go suporta ponteiros , permitindo que você passe referências a valores e registros dentro do seu programa. Mostraremos como os ponteiros funcionam em contraste com os valores com 2 funções: zeroval e zeroptr. zeroval tem um intparâmetro, então os argumentos serão passados para ele por valor. zeroval obterá uma cópia ival distinta daquela na função de chamada. zeroptr em contraste tem um *intparâmetro, o que significa que leva um intponteiro. O *iptr código no corpo da função então desreferencia o ponteiro de seu endereço de memória para o valor atual nesse endereço. Atribuir um valor a um ponteiro sem referência altera o valor no endereço referenciado.

```go
package main

import "fmt"

func zeroval(ival int) {
    ival = 0
}



func zeroptr(iptr *int) {
    *iptr = 0
}




func main() {
    i := 1
    fmt.Println("initial:", i)

    zeroval(i)
    fmt.Println("zeroval:", i)

    zeroptr(&i)
    fmt.Println("zeroptr:", i)

    fmt.Println("pointer:", &i)
}
```

1.8. Recursion

Go suporta funções recursivas . Aqui está um exemplo clássico. Essa fact função chama a si mesma até atingir o caso base de fact(0). Closures também podem ser recursivos, mas isso requer que o closure seja declarado com um tipo varexplicitamente antes de ser definido.

```
package main

import "fmt"

func fact(n int) int {
    if n == 0 {
        return 1
    }
    return n * fact(n-1)
}

func main() {
    fmt.Println(fact(7))

    var fib func(n int) int



    fib = func(n int) int {
        if n < 2 {
            return n
        }

        return fib(n-1) + fib(n-2)
    }

    fmt.Println(fib(7))
}
```

1.9. Strings and Runes

Uma string Go é uma fatia de bytes somente leitura. A linguagem e a biblioteca padrão tratam as strings especialmente - como contêineres de texto codificado em UTF-8 . Em outras linguagens, as strings são feitas de “caracteres”. Em Go, o conceito de caractere é chamado de a rune- é um número inteiro que representa um ponto de código Unicode. Esta postagem do blog Go é uma boa introdução ao tópico.

```go
package main

import (
    "fmt"
    "unicode/utf8"
)

func main() {

    const s = "สวัสดี"



    fmt.Println("Len:", len(s))


    for i := 0; i < len(s); i++ {
        fmt.Printf("%x ", s[i])
    }
    fmt.Println()

    fmt.Println("Rune count:", utf8.RuneCountInString(s))







    for idx, runeValue := range s {
        fmt.Printf("%#U starts at %d\n", runeValue, idx)
    }

    fmt.Println("\nUsing DecodeRuneInString")
    for i, w := 0, 0; i < len(s); i += w {
        runeValue, width := utf8.DecodeRuneInString(s[i:])
        fmt.Printf("%#U starts at %d\n", runeValue, i)
        w = width

        examineRune(runeValue)
    }
}

func examineRune(r rune) {

    if r == 't' {
        fmt.Println("found tee")
    } else if r == 'ส' {
        fmt.Println("found so sua")
    }
}
```

1.10. switch/case

As instruções switch expressam condicionais em muitas ramificações. Você pode usar vírgulas para separar várias expressões na mesma caseinstrução. Usamos o default caso opcional neste exemplo também. switch sem uma expressão é uma maneira alternativa de expressar a lógica if/else. Aqui também mostramos como as case expressões podem ser não constantes. Um tipo switch compara tipos em vez de valores. Você pode usar isso para descobrir o tipo de um valor de interface. Neste exemplo, a variável t terá o tipo correspondente à sua cláusula. Aqui está um básico switch.

```
package main

import (
    "fmt"
    "time"
)

func main() {

    i := 2
    fmt.Print("Write ", i, " as ")
    switch i {
    case 1:
        fmt.Println("one")
    case 2:
        fmt.Println("two")
    case 3:
        fmt.Println("three")
    }

    switch time.Now().Weekday() {
    case time.Saturday, time.Sunday:
        fmt.Println("It's the weekend")
    default:
        fmt.Println("It's a weekday")
    }

    t := time.Now()
    switch {
    case t.Hour() < 12:
        fmt.Println("It's before noon")
    default:
        fmt.Println("It's after noon")
    }

    whatAmI := func(i interface{}) {
        switch t := i.(type) {
        case bool:
            fmt.Println("I'm a bool")
        case int:
            fmt.Println("I'm an int")
        default:
            fmt.Printf("Don't know type %T\n", t)
        }
    }
    whatAmI(true)
    whatAmI(1)
    whatAmI("hey")
}
```

2. References

Sites que foram usados de base foi: [gobyexample 2022] e [Programiz 2022]

References

gobyexample (2022). Go by example. https://gobyexample.com. Accessed: 2022-04-06.

Programiz (2022). Go for loop. https://www.programiz.com/golang/for-loop. Accessed: 2022-04-06.