

# Alati za razvoj softvera

Programski jezik Go (golang) — nastavak



Univerzitet u Novom Sadu  
Fakultet Tehničkih Nauka

## Pokazivači

- ▶ Slično kao u C programskom jeziku
- ▶ Tip podatka koji kao vrednost drži adresu
- ▶ Za razliku od jezika C, Go nema pokazivačku aritmetiku
- ▶ Ako hoćemo da promenimo vrednost nekog elementa, moramo ga proslediti **po adresi** tj. preko pokazivača
- ▶ Ako prosledimo bez pokazivača dobijamo kopiju, tj. original neće biti izmenjen!
- ▶ Prenos preko pokazivača (adrese) je znatno brži od prenosa po vrednosti zato što nema kopiranja vrednosti

```
/*  
- * je pokazivac  
- & je adresa  
*/  
func pointersExample() {  
    i := 42  
    p := &i // pokazivac na i tj. u p smestamo adresu od i  
    fmt.Println(*p) // deferenciranje pokazivaca, uzimamo vrednost  
    *p = 21 // postavljanje vrednosti i  
    fmt.Println(*p)  
    fmt.Println(i) // ispisi i  
}
```

# Strukture

- ▶ Slično kao u C programskom jeziku
- ▶ Koriste se za modelovanje složenih tipova podataka
- ▶ Slično kao klasa, samo što **nije** klasa
- ▶ Možemo *zakačiti* funkcije na njih, ali to **nisu** metode
- ▶ Ako hoćemo da promenimo stanje strukture moramo to raditi preko pokazivača

```
package main

import "fmt"

type Person struct {
    firstName string
    lastName  string
}

func (p Person) SayHi() string {
    return p.firstName + " hi"
}

func (p *Person) SetName(name string) {
    p.firstName = name
}

func main() {
    p := Person{"Milos", "Simic"}
    fmt.Println(p.SayHi())
    p.SetName("Jovan")
    fmt.Println(p.SayHi())
}
```

## Prava pristupa

- ▶ Go **nije** objektno-orijentisani jezik
- ▶ Stoga, skrivanje informacija (atributa) kroz nivoe pristupa (public, protected, private) nije moguće
- ▶ Nešto slično kao i python, Go pruža mogućnost da **ipak** zatvorimo/otvorimo kako attributed, tako i funkcije
- ▶ Jedino pravilo koje ovde postoji je sledeće:
  1. Atributu ili funkciji koja počinje velikim slovom, možemo pristupiti i van strukture

- ▶ Go ide jedan korak dalje...prethodno pravilo važi i za bilo koju funkciju u bilo kom paketu
- ▶ Funkcije mogu da postoje nezaivсно od neke strukture
- ▶ Ako želimo da im pristupimo u nekom drugom fajlu/paketu moramo ih **eksportovati**
- ▶ Moramo naziv funkcije početi sa velikim slovom

## Statici i dinamički nizovi

- ▶ Kao i u drguim programskim jezicima, možemo praviti kolekcije elemenata
- ▶ Kao elemetn kolekcije, možemo da izaberemo bilo koji prost ili složen tip
- ▶ Ove kolekcije mogu biti statičke (veličina unapred poznata), ili dinamičke (veličina unapred nije poznata)



```
// kreiramo staticki niz duzine 2
var names [2]string

// na poziciju 0 stavljamo "Marc", na poziciju 1 stavljamo "John"
names[0] = "Marc"
names[1] = "John "

// Ispisujemo elemente na poziciji 0 i 1
fmt.Println(names[0], names[1])

// Ispisujemo ceo niz
fmt.Println(names)
```

- ▶ Statički nizovi nisu uvek pogodni jer se veličina mora znati unapred
- ▶ Zbog toga se koriste dinamični nizovi, *slice* - identičan listi u drugim programskim jezicima
- ▶ Slično se ponaša i slične operacije možemo izvoditi

```
slice := []int{}  
fmt.Println(slice) // []  
slice = append(slice, 3) // dodajemo broj 3 u slice  
fmt.Println(slice) // [3]
```

# Mape

- ▶ Go podržava rad sa strukturama ključ:vrednost
- ▶ U nekim jezicima se ovo zove mapa, u nekim dict
- ▶ Ključ treba da bude string, vrednost može biti bilo šta
- ▶ Ponaša se identivno kao sličan strukture u drugim jezicima

```
m = make(map[string]int) // ili ovako m = map[string]int{}
m["route"] = 66

i := m["route"] // 66
n := len(m) // 1

delete(m, "route") // brisanje po kljucu

i, ok := m["route"] // cest patern, proverava da li kljuc postoji ili ne

for key, value := range m { // iteriranje kroz mapu
    fmt.Println("Key:", key, "Value:", value)
}
```

# Interfejsi

- ▶ Interfejsi predstavljaju imenovane kolekcije potpisa metoda
- ▶ Analogni su interfejsima u drugim jezicima
- ▶ Dodavanje nečg nalik na objektne paradigme na Golang
- ▶ Ključna reč *interface*
- ▶ Interfejs se ne može da naslediti dugi interfejs
- ▶ ALI moguće je je kombinovati interfejse i od dva interfejsa napraviti novi koji sadrči sve funkcije intejfejsa od kojih se sastoji
- ▶ Omogućavaju duck typing, (odnosno structural typing)
- ▶ Ako neka struktura implementira sve operacije interfejsa, implementira interfejs

```
package main

import "fmt"

type Osoba interface {
    ToString() string
}

type Student struct {
    ime, prz, brIndeksa string
}

type Radnik struct {
    ime, prz, jmbg string
}

func (s Student) ToString() string {
    return "Student[" + s.ime + " , " + s.prz + " ," + s.brIndeksa + "]"
}

func (r Radnik) ToString() string {
    return "Radnik[" + r.ime + " , " + r.prz + " ," + r.jmbg + "]"
}
```

# Nastavak

```
func display(o Osoba) {  
    fmt.Println(o.ToString())  
}  
  
func main() {  
    s := Student{"marko", "markovic", "ee-222/2012"}  
    r := Radnik{"rastko", "raicevic", "055121312321312"}  
  
    display(s)  
    display(r)  
}
```



## Konkurentnost — osnove

- ▶ Funkcije se mogu izvršavati konkurentno sa drugim funkcijama koristeći go rutine (goroutines)
- ▶ Gorutine su lakše od niti
- ▶ Sinronizacija gorutina se vrši pomoću kanala
- ▶ Pozivaju se ključnom rečju **go**
- ▶ Main funkcija se izvršava u sopstevnoj gorutini
- ▶ Pokretanje gorutine odmah vrać a kontrolu pozivaocu:
  - ▶ Ne šeka se kraj izvršavanja gorutine
  - ▶ Sve povratne vrednosti gorutine se ignorišu
- ▶ Main gorutina mora biti pokrenuta da bi se bilo koja druga gorutina izvršavala
- ▶ Prekid main gorutin, prekida sve ostale gorutine — leaks

```
func hello(from string) {  
    for i := 1; i < 100000000; i++ {  
    }  
    fmt.Println("Hello from : " + from)  
}  
  
func main() {  
    hello("program")  
    go hello("Go routine")  
    go func(caller string) {  
        fmt.Println("Anonymous f: called by " + caller)  
    }("Go routine")  
    fmt.Scanln()  
}
```

## Kanali — osnove

- ▶ Kanali predstavlja mehanizam pomoću kojeg komuniciraju Gorutine
- ▶ Kanal u suštini predstavlja strukturu koja funkcioniše po FIFO principu (nešto poput reda)
- ▶ Svaki kanal ima tip podatka koji mu je pridružen
- ▶ To ograničava kanal na tip podatka kojim se može komunicirati unutar njega

```
\\kreiranje kanala tipa int  
a := make(chan int)
```

```
\\citanje iz kanala a <- data - pisanje u kanal  
data := <- a
```

# Paketi

- ▶ Za razliku od većine drugih programskih jezika, go nam omogućava da grupišemo stvari
- ▶ Tipičan primer gde bi ovo koristili je **import** deo
- ▶ Ne moramo pisati **import** kod svakog paketa možemo ih spojiti

```
\\Grupisani paketi
import (
    "fmt"
    "math"
)
```

```
\\Razdvojeni paketi
import "fmt"
import "math"
```

## Ugnježdeni i imenovani paketi

- ▶ Kao i drugi programski jezici, go omogućava da koristimo ugnježdene importe
- ▶ Ovo nije ništa spektakularno, samo znači da možemo importovati podpaket nekog paketa
- ▶ Druga zgodna osobona koju možemo da iskoristimo a liči na neke mogućnosti pajtona su imenovani paketi
- ▶ Ako imate importe koji se zovu isto, to može dovesti do problema
- ▶ Ovim paketima možemo da ti **alias** i pristupati im tako

```
\\Ugnjezdeni paket
```

```
package main
```

```
import (  
    "fmt"  
    "math/rand"  
)
```

```
func main() {  
    fmt.Println(rand.Int(100))  
}
```

```
\\ Alias paket
```

```
import (  
    f "fmt"  
    m "math"  
)
```

```
func main() {  
    c := m.Exp2(5)  
    f.Println(c)  
}
```

## Go mod

- ▶ Go nam omogućava da instaliramo dodane pakete u naše projekte
- ▶ Ideja kao i u bilo kom drugom programskom jeziku
- ▶ Go podržava nekoliko alata za to, a neki dolaze direktno sa instalaciom go-a
- ▶ Za razliku od pajtona, nema potrebe da se prave virtuelna okruženja
- ▶ Trebate da napravite folder gde će biti vaš projekat i da pozovete par jednostavnih komandi
- ▶ Dobra stvar je što prilikom testiranja ili kompajliranja ili pokretanja vašeg programa, alat proverava pakete i instalira šta nedostaje

1) Kreiranje foldera

```
$ mkdir projects
```

```
$ cd projects
```

```
$ mkdir mymodule
```

2) Struktura foldera

```
|--- projects
```

```
    |--- mymodule
```

3) Inicijalizacija `go` mod alata

```
go mod init mymodule
```

Output

```
go: creating new go.mod: module mymodule
```

4) truktura projekta

```
|--- projects
```

```
    |--- mymodule
```

```
        |--- go.mod
```

```
        |--- main.go
```

5) Instalacija spoljnih biblioteka

```
go get github.com/spf13/cobra
```



## Dodatni materijali

- ▶ Tour of Go
- ▶ go mod
- ▶ Malo više o kanalima i gorutinama
- ▶ Razlika niti i gorutina
- ▶ Razlika procesi i niti
- ▶ The Go Programming Language

# Kraj predavanja

Pitanja? :)