Napredni algoritmi i strukture podataka

Programski jezik Go

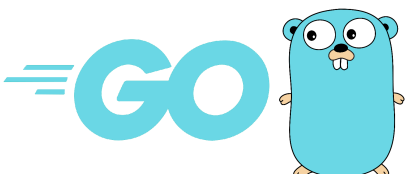
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Osnovne informacije I

- Go je programski jezik otvorenog koda razvijen 2009. godine od strane Google-a, odnosno od strane Robert Griesemer-a, Rob Pike-a i Ken Thompson-a
- Ima velike sličnosti sa C programksih jezikom, ali poseduje memory safety i garbage collection mehanizme, pa ga mnogi neformalno nazivaju "modernim C-om"
- Go ima 3 bitne karakteristike koje ga razlikuje od ostalih programksihg jezika:
  1. **Jednostavan dizajn** — ne postoji mehanizam rukovanje izuzecima niti generic templates (do verzije 1.18, kada je dodata minimalna podrška za generics) koji komplikuju dizajn samog programskog jezika
  2. **Direktna podrška za konkurentno programiranje** — poseduje rutine (goroutines) koji omogućuvaju konkuretno izvršavanje funkcija bez potrebe da se koristi bilo kakva eksterna biblioteka (npr. pthreads u C-u)
  3. **Backward compatibility** — izmene koje bi narušile kompatibilnost prethodnih verzija se neće uvoditi u sam programki jezik

Osnovne informacije II

- Uputstvo za instalaciju Go-a nalazi se na sledećem linku
- Komande za kompajliranje i pokretanje programa:
  - **go build** [go file/import path/file system path] - Kompajlira program i u slučaju main paketa čuva executable fajl u trenutnom radnom direktorijumu
  - **go install** [packages] - Kompajlira program i u slučaju main paketa čuva executable fajl u $GOBIN direktorijumu. Pre toga radi download sa repozitorijuma ako je potrebno.
  - **go run** [go file/import path/file system path] - Kompajlira i pokreće program, binary se čuva u privremenom direktorijumu i briše se nakon izvršavanja programa



Hello world

```
package main // deklaracija paketa

import "fmt" // paket za standardni ulaz / izlaz

// ulazna tacka programa
func main () {
        fmt.Println("Hello World from Go")
}
```

Organizacija koda

- Program se sastoji iz jednog ili više fajlova sa ekstenzijom **.go**
- Svi go fajlovi unutar jednog direktorijuma pripadaju istom **paketu**
- Naziv paketa mora se navesti na početku svakog fajla
- Go poseduje mehanizam da različite konstrukte u kodu učinimo dostupnim ili nedostupnim van paketa u kom se nalaze
- Eksportovani konstrukti (promenljive, funkcije konstante itd). započinju velikim slovom i mogu se referencirati iz bilo kog paketa
- Neeksportovani elementi mogu se referencirati samo iz paketa u kom se nalaze, van njega nisu vidljivi
- Kako bi se konstrukti definisani unutar jednog paketa mogli koristiti unutar nekog drugog paketa, naziv tog paketa mora se navesti unutar **import** sekcije

Promenljive I

- Mogu se deklarisati i inicijalizovati na nivou paketa ili na nivou funkcije
- Nazivi promenljivih i njihovih tipova navode se nakon ključne reči **var**, nakon čega se mogu i inicijalizovati
- Go poseduje type inference mehanizam, što znači da se tip promenljive ne mora navesti prilikom inicijalizacije (kompajler može sam da zaključi)
- Alternativni način inicijalizacije promenljive na nivou funkcije (skraćeni oblik) je pomoću := operatora

Promenljive II

```go
package main

import "fmt"

var a int
var b, c string = "hello", "world"
var d, e = 1, false

func main() {
	var i int
	var j = 5
	k := true

	//promenljive definisane na nivou paketa
	fmt.Println(a, b, c, d, e)
	//output: 0 hello world 1 false

	//promenljive definisane na nivou funkcije
	fmt.Println(i, j, k)
	//output: 0 5 true

	fmt.Printf("type of k is: %T", k)
	//output: type of k is: bool
}
```

Konstante

- Rad sa konstantama isti je kao rad sa varijablama, uz zamenu ključne reči var rečju **const**
- Skraćeni oblik inicijalizacije nije moguće primeniti na konstante
- Kada je potrebno uzastopno inicijalizovati više promenljivih ili konstanti, deklaracije je moguće grupisati

```
const (
      Pi = 3.14
      E = 2.72
)
```

- Grupisanje importa paketa obavlja se na isti način

Primitivni tipovi podataka

- bool
- string
- int, int8, int16, int32 (rune), int64
- uint, uint8 (byte), uint16, uint32, uint64, uintptr
- float32, float64
- complex64, complex128
- Pri operaciji dodele potrebno je vršiti eksplicitnu konverziju tipova

```
var i int = 5
j := i // ok, j je sada int
//var k float32 = i - ne radi
var k float32 = float32(i)
```

Funkcije I

- Opšti oblik funkcije

```
func naziv_funkcije(param1 tip, param2 tip ...) (rv1 tip, rv2 tip ..) {
      //telo funkcije
}
```

- Ukoliko više uzastopnih parametara ima isti tip, on se može navesti samo iza naziva poslednjeg parametra

```
func add(x, y int) int {
      return x + y
}
```

Funkcije II

- Funkcija ne mora imati nijednu povratnu vrednost, a može ih imati i više

```
func swap(x, y int) (int, int) {
      return y, x
}
```

- Povratne vrednosti mogu biti imenovane (preporučljivo samo za kratke funkcije)

```
func add(x, y int) (sum int) {
      sum = x + y
      return
}
```

Rukovanje greškama

- Go ne poseduje mehanizam za rukovaneje izuzecima
- Kao poslednja povratna vrednost funkcije najčešće se prosleđuje indikator greške koji treba da bude promenljiva čiji tip implementira interfejs **error**
- Kod koji poziva tu funkciju treba da obradi grešku na odgovarajući način

```
func main() {
      i, err := strconv.Atoi("42")
      if err != nil {
        fmt.Printf("couldn't convert number: %v\n", err)
        return
      }
      fmt.Println("Converted integer:", i)
}
```

- **Napomena** - null vrednost u Go-u naziva se nil

For I

- For je jedini konstrukt u Go-u koji predstavlja petlju
- Osnovni oblik for petlje podrazumeva tri komponente:
  - Inicijalni iskaz (uglavnom deklaracija promenljive)
  - Uslovni izraz koji se evaluira pre svake iteracije
  - Iskaz koji se izvršava na kraju svake iteracije

```
for i := 1; i > 10; i++ {
      fmt.Println(i)
}
```

- Inicijalni i krajnji iskaz su opcioni i mogu se izostaviti

```
i := 1
//petlja bez pocetnog i krajnjeg iskaza
for ; i < 10 ; {
        fmt.Println(i)
      i += 1
}
```

For II

- Iako while petlja ne postoji, ona se može predstaviti for petljom koja sadrži samo uslovni izraz

```
i := 1
//imitacija while petlje
for i < 100 {
        fmt.Println(i)
      i += 1
}
```

- Petlja gde su sve tri komponente izostavljene predstavlja beskonačnu petlju
- Komponente for petlje nije neophodno stavljati u male zagrade, dok su velike zagrade nakon toga obavezne

If-Else

- If izraz ne mora biti unutar malih zagrada, dok su velike zagrade nakontoga obavezne
- else ključna reč mora se navesti u istoj liniji kao i } iz if bloka
- Pre navođenja uslovnog izraza moguće je navesti iskaz i na taj način deklarisane promenljive vidljive su samo unutar if bloka ili bilo koje njegove else grane

```
func examPassed(points int, maxPoints int) bool {
      if percentage := points / maxPoints * 100; percentage >= 51 {
        return true
      } else {
        return false
      }
}
```

Switch-Case

- Slična je onima u drugim programskim jezicima
- Slučajevi se proveravaju od vrha ka dnu i samo prvi pogodak na koji se naiđe biće izvršen, tako da nije potrebno pisati break iskaze
- case izrazi ne moraju biti konstante

```
func main() {
      fmt.Println("When's Saturday?")
      today := time.Now().Weekday()
      switch time.Saturday {
        case today + 0:
          fmt.Println("Today.")
        case today + 1:
          fmt.Println("Tomorrow.")
        default:
          fmt.Println("Too far away.")
      }
}
```

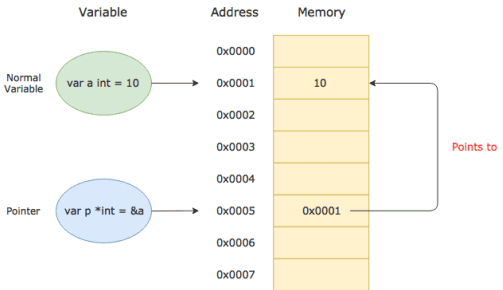
Defer

- Odlaže izvršavanje navedene funkcije do trenutka završetka funkcije u kojoj se naredba nalazi
- Argumenti navedene funkcije se odmah evaluiraju, ali se sam poziv odlaže
- Funkcija može sadržati više defer naredbi i one se izvršavaju u LIFO redosledu

```
func main() {
      fmt.Println("counting")
      for i := 0; i < 10; i++ {
        defer fmt.Println(i)
      }
      fmt.Println("done")
      // output:
      // counting
      // done
      // 9
      // ...
      // 0
}
```

Pokazivači I

- Pokazivači su promenljive čije vrednosti su adrese drugih promenljivih
- Pokazivač tipa *T sadrži adresu memorijske lokacije u kojoj se nalazi (ili započinje) neka vrednost tipa T



Pokazivači II

- Dobavljanje vrednosti na koju pokazivač pokazuje naziva se dereferenciranje pokazivača i za to služi operator *
- Operator & kreira pokazivač na zadati operand tako što vraća adresu na kojoj se vrednost promenljive čuva

```
i := 5
var p *int = &i // u pokazivač p upisuje se adresa
                // memorijske lokacije na kojoj se nalazi i
fmt.Println(p)
// output: 0xc00000a098
fmt.Println(*p)
// output: 5
*p += 5
fmt.Println(*p)
//output: 10
```

(Statički) nizovi

- Niz predstavlja strukturu fiksne dužine i čuva kolekciju vrednosti istog tipa
- Prilikom deklaracije potrebno je navesti dužinu niza

```
var a [2]string
a[0] = "Hello"
a[1] = "World"
fmt.Println(a[0], a[1])
//output: Hello World
fmt.Println(a)
//output: [Hello World]
```

- Na pozicijama gde elementi nisu inicijalizovani, prostor se popunjava nultim vrednostima za taj tip podataka

```
nums := [6]int{1, 2, 3, 4}
fmt.Println(nums)
//output: 1 2 3 4 0 0
```

Dinamički nizovi (Slice) I

- Za razliku od niza koji ima fiksan broj elemenata, određen pri deklaraciji promenljive, slice je promenljive dužine
- Slice predstavlja referencu na neki statički niz, i ta referenca se menja po potrebi, kada slice raste ili se smanjuje
- Kada više od jednog slice-a referencira isti niz, izmene nad njim vidljvie su iz svakog tog slice-a
- Jedan od načina za kreiranje slice-a je pomoću make funkcije, kojoj se zadaju inicijalna dužina i opciono kapacitet

```
a := make([]int, 5, 5)
printSlice("a", a)
//output: a len=5 cap=5 [0 0 0 0 0]

b := make([]int, 0, 5)
printSlice("b", b)
//output: b len=0 cap=5 []
```

Dinamički nizovi (Slice) II

- Drugi način je kreiranje na osnovu postojećeg statičkog ili dinamičkog niza

```
c := b[:2]
printSlice("c", c)
//output: c len=2 cap=5 [0 0]

d := c[2:5]
printSlice("d", d)
//output: d len=3 cap=3 [0 0 0]
```

Dinamički nizovi (Slice) III

- Iteriranje kroz niz ili slice vrši se upotrebom range oblika for petlje gde se pri svakoj iteraciji vraćaju dve vrednosti: trenutni indeks i vrednost elementa na tom indeksu

```
var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

func main() {
      for i, v := range pow {
        fmt.Printf("2**%d = %d\n", i, v)
      }
      // output:
      // 2**0 = 1
      // 2**1 = 2
      // ...
}
```

Mape I

- Mapa predstavlja kolekciju parova ključ-vrednost
- Kako bi se elementi mogli dodavati, potrebno je prvo inicijalizovati mapu uz pomoć funkcije make

```
m := make(map[string]int)

m["Answer"] = 42
fmt.Println("The value:", m["Answer"])
//output: The value: 42

m["Answer"] = 48
fmt.Println("The value:", m["Answer"])
//output: The value: 48
```

Mape II

- Ako ključ nije pristuan u mapi vratiće se nulta vrednost za taj tip

```
delete(m, "Answer")
fmt.Println("The value:", m["Answer"])
//output: The value: 0

v, ok := m["Answer"]
fmt.Println("The value:", v, "Present?", ok)
//output: The value: 0 Present? false
```

Strukture I

- Struktura predstavlja kolekciju polja, svako polje je promenljiva određenog naziva i tipa
- Strukture modeluju stanja objekata

```
type Vertex struct {
      X, Y float64
}
```

- Klase iz OO programiranja i strukture u Go-u nisu ekvivalenti

Strukture II

- Ukoliko je promenljiva pokazivač na neku strukturu, nije neophodno vršiti eksplicitno dereferenciranje tog pokazivača

```
func main() {
      var (
        v1 = Vertex{1, 2}
        v2 = Vertex{X: 1}
        v3 = Vertex{}
        p *Vertex = &Vertex{1, 2}
      )
      fmt.Println(v1, p.Y, v2, v3)
      //output: {1 2} 2 {1 0} {0 0}
}
```

Metode I

- Go nije OO jezik i ne podržava klase, međutim nad strukturama i imenovanim tipovima mogu se kreirati metode
- Metode predstavljaju funkcije koje poseduju specijalni receiver argument koji se navodi između ključne reči func i naziva funkcije
- Ovakva sintaksa samo je syntactic sugar za pisanje funkcija čiji je prvi parametar tipa neke strukture
- Metoda se može definisati samo nad tipom koji se nalazi u istom paketu

Metode II

```
func (v Vertex) Abs() float64 {
	return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func main() {
	v := Vertex{3, 4}
	fmt.Println(v.Abs())
	//output: 5
}
```

- Kada se metoda poziva nad promenljivom nekog tipa, vrednost te promenljive se kopira i bilo kakve izmene nad vrednošću unutar metode neće se reflektovati na promenljivu nad kojom je metoda pozvana

Metode III

- Metode koje menjaju vrednosti promenljive potrebno je pisati nad pokazivačima, jer se u tom slučaju kopira samo adresa i izmene su vidljive

```
func (v *Vertex) Scale(f float64) {
	v.X = v.X * f
	v.Y = v.Y * f
}

func main() {
	v := Vertex{3, 4}
	v.Scale(10)
	fmt.Println(v.Abs())
	//output: 50
}
```

Interfejsi I

- Interfejs predstavlja kolekciju potpisa metoda
- Bilo koja vrednosti čiji tip implementira sve metode interfejsa može se konvertovati u tip tog interfejsa
- Implementacija metoda interfejsa je implicitna, podrazumeva se da tip implementira interfejs ukoliko implementira sve njegove metode

```
type Printer interface {
      Print()
}

type Greeter struct {
      name string
}

. . .
```

Interfejsi II

```
. . .

func (g Greeter) Print() {
	fmt.Printf("Hi %s!", g.name)
}

func main() {
	var greeterPrinter Printer = Greeter{"Bob"}
	greeterPrinter.Print()
	//output: Hi Bob!
}
```

Rad sa modulima

- Go upravlja zavisnostima pomoću modula
- Modul se sastoji iz jednog ili više paketa i u korenu sadrži go.mod fajl
- Na početku go.mod fajla navodi se module path, koji treba jedinstveno da identifikuje modul
- Nakon toga, navodi se verzija Go-a, iza čega slede **require** direktive koje navode zavisnosti tog modula
- **replace** direktiva menja sadržaj modula sadržajem koji se nalazi na navedenoj putanji (korisno u situacijama kada koristite module koji nigde nisu objavljeni)

```
module a

go 1.17

require rsc.io/quote v1.5.2
```

Komande za upravljanje zavisnostima

- **go mod init** module_path - Kreira modul tako što dodaje go.mod fajl u trenutnom radnom direktorijumu
- **go get** module_path@version - Navodi zavisnost u go.mod fajlu i dodaje je u module cache ukoliko ona tamo već ne postoji
- **go mod download** [module_path] - Ako se module path ne navede, u module cache dodaje zavisnosti navedene u trenutnom go.mod fajlu, a ako se module path navede, dodaje samo navedeni modul u module cache, ali bez izmene go.mod fajla
- **go mod tidy** - Ažurira go.mod fajl tako da zavisnosti odgovaraju onima koje se nalaze u izvornom kodu i dodaje module u module cache ukoliko nisu prisutni tamo

Podrška za konkurentno programiranje

- Konkurentno programiranje omogućava isprepletano izvršavanje više tokova programa (niti)
- Go ima bogatu podršku za konkurentno programiranje
- Goroutines su lightweight niti kojima upravlja Go runtime, a ne OS
- Go rutine mogu komunicirati preko kanala, bez potrebe za zaključavanjem deljene memorije

```
func say(s string) {
        for i := 0; i < 5; i++ {
                time.Sleep(100 * time.Millisecond)
                fmt.Println(s)
        }
}

func main() {
        go say("world")
        say("hello")
}
```

Zadaci I

1. Napisati funkciju koja proverava da li je zadati broj manji, veći ili jednak nuli i ispisati rezultat na konzoli
   - Koristeći if-else
   - Koristeći switch-case
2. Kreirati slice integer-a i iteriranjem uz pomoć for-range petlje izdvojiti elemente čija je vrednost manja od njihovog indeksa
3. Napisati funkciju koja za zadati ulaz n vraća n-ti prost broj. Druga povratna vrednost funkcije treba da bude indikator greške. Pravilno obraditi slučaj kada je vrednost argumenta funkcije manja od 1. (Više o radu sa greškama)
4. Kreirati mapu i popuniti je vrednostima tako da ključ predstavlja poštanski broj, a vrednost naziv grada. Nakon toga prolaskom kroz mapu izdvojite sve jedinstvene nazive gradova, sačuvajte ih u sekvenci i ispišite na konzolu.

Zadaci II

5. Rad sa strukturama:
   - Kreirati strukturu Student sa poljima: ime, prezime, broj indeksa
   - Napraviti promenljivu tipa Student koja sadrži vaše lične podatke
   - Ispišite vrednost promenljive na konzolu
   - Kreirajte metodu za izmenu broja indeksa i primenite je nad kreiranom promenljivom
   - Ponovo ispišite vrednost promenljive kako biste se uverili da se izmena desila
6. Napisati funkciju koja proverava da li je zadati broj Armstrongov broj (Objašnjenje Armstrongovog broja).
7. Napisati funkciju sa parametrima word (string) i candidates (sekvenca stringova). Potrebno je proveriti koji stringovi iz date sekvence predstavljaju anagram parametra word. Povratna vrednost funkcije treba da bude sekvenca pronađenih anagrama. Testirati svoje rešenje za sledeći ulaz:
   - word: listen
   - candidates: enlists, google, inlets, banana
   - rešenje: inlets

Zadaci III

8. Implementirati jednostruko spregnutu listu:
   - Struktura LinkedList predstavlja listu celobrojnih vrednosti
   - Interfejs List nudi metode za:
     - Dodavanje novog elementa na zadatom indeksu i
     - Brisanje elementa na zadatom indeksu i
     - Proveru postojanja elementa sa vrednošću x
   - Struktura LinkedList treba da implementira sve metode interfejsa List tako da se operacije dodavanja i brisanja odraze na samu strukturu
   - Voditi računa o greškama koje se mogu desiti prilikom dodavanja ili brisanja elementa i pravilno ih obraditi

Dodatni materijali

- Goland By Example
- Learn X in Y minutes Where X=Go
- Go by Example
- Golang Tutorial Series
- Go Programming
- Concurrency
- Golang Concurrency
- Rob Pike - 'Concurrency Is Not Parallelism'