Programske paradigme

— Programski jezik Go —

Milena Vujošević Janičić

Matematički fakultet, Univerzitet u Beogradu

Sadržaj

1 Uvod

Početak razvoja

Google 2007

- Programski jezik Go je projekat koji razvija kompanija Google od 2007. godine.
- Postao je javno dostupan kao projekat otvorenog koda 2009. godine i u stalnom je razvoju.
- **Cilj projekta:** kreirati novi statički tipiziran jezik koji se kompilira, a koji omogućava jednostavno programiranje kao kod interpretiranih, dinamički tipizarnih programskih jezika.

Osobine

C familija programskih jezika

- Smatra se da Go pripada C familiji programskih jezika, ali pozajmljuje i adaptira ideje raznih drugih jezika, izbegavajući karakteristike koje dovode do komplikovanog i nepouzdanog koda.
- Go nije objektno-orijentisan, ali podržava određene koncepte kao što su metodi i interfejsi koji pružaju fleksibilnu apstrakciju podataka.
- Go omogućava efikasnu konkurentnost koja je ugrađena u sam jezik, kao i automatsko upravljanje memorijom, odnosno sakupljanje smeća.

Osobine

Primene

- Zbog svojih osobina kao što je konkurentnost, Go je posebno dobar za izradu različitih vrsta serverskih aplikacija, ali je pre svega jezik opšte namene pa se može koristiti za rešavanje svih vrsta problema.
- Ima primenu u raznim oblastima kao što su grafika, mobilne aplikacije, mašinsko učenje i još mnoge druge.
- Osim kompanije Google, koristi se u velikom broju drugih kompanija kao alternativa jezicima poput Python-a i Javascript-a, jer pruža znatno viši stepen efikasnosti i bezbednosti.

2 Osnovne karakteristike

Alat `go`

Upravljanje kodom
Alat `go` predstavalja osnovni alat za upravljanje kodom napisanim u programskom jeziku Go. Neke od definisanih komandi u okviru alata su:

- `build` za kompilaciju paketa,
- `run` za kompilaciju i izvršavanje Go programa,
- `test` za testiranje paketa,
- `get` za preuzimanje i instaliranje paketa,
- `fmt` za formatiranje koda paketa.

Upotreba alata: `go komanda [argumenti]`

Hello world

Paketi, komande, komentari

- Na početku programa navodi se naziv paketa. Svaki paket koji sadrži `main` funkciju mora nositi naziv `main`.
- Sa naredbom `import` navode se nazivi svih paketa koji se koriste u programu. Go ne dozvoljava uključivanje suvišnih paketa, već se mogu navesti samo paketi koji se upotrebljavaju u programu. Ukoliko se navede paket koji se ne koristi, biće prijavljena greška tokom kompilacije.
- Nakon svake naredbe nije obavezno navođenje znaka `;` , osim ako je potrebno navesti više naredbi u istoj liniji.
- Komentari se navode na isti način kao u jeziku C.

Hello world

Hello world

Primer 1: Program *Hello world!* u jeziku Go

```
package main

import "fmt"

func main() {
    fmt.Println("Hello world!") // ispisuje Hello world!
}
```

Paketi

Standardna biblioteka

U okviru same distribucije programskog jezika Go postoji veliki broj paketa standardne biblioteke koji obezbeđuju različite usluge:

- `fmt` za formatirani ulaz/izlaz;
- `strings` za rad sa stringovima;
- `math` sa implementacijama matematičkih funkcija.
- `os` koji predstavlja interfejs ka operativnom sistemu i omogućava funkcionalnosti kao što su rad sa sistemom datoteka, pristup argumentima komandne linije i pokretanje procesa.

U standardnoj biblioteci takođe postoje paketi za rad sa SQL bazama podataka, kriptografskim funkcijama, za kompresiju podataka, rad sa slikama, podršku za mrežno programiranje...

Paketi

Dodatni paketi

Pored paketa standardne biblioteke postoje dodatni i prošireni paketi u okviru Go projekta koji imaju labavije uslove kompatibilnosti Go verzija. Ovde se mogu pronaći paketi za razvoj aplikacija za mobilne platforme, eksperimentalni debager, prošireni kriptografski i mrežni paketi.

Paketi

Dodatni paketi

- Paketi van distribucije samog jezika mogu se preuzeti korišćenjem komande `go get` i navođenjem repozitorijuma u kome se paket nalazi.
- Na veb sajtu https://godoc.org/ mogu se naći paketi i njihove dokumentacije koje razvijaju članovi Go zajednice koja vremenom postaje sve veća.
- Paketi su odobreni od strane Go projekta i mogu se naći paketi koji obezbeđuju najrazličitije alate za programiranje kao i API-je mnogih postojećih servisa.

Tipovi podataka

Osnovni, složeni, referentni, interfejsni

Tipovi podataka koji su definisani u programskom jeziku Go mogu se klasifikovati u četiri kategorije:

1. Osnovni tipovi (numerički, bulovski, tekstualni)
2. Složeni tipovi (nizovi i strukture)
3. Referentni tipovi (pokazivači, isečci, mape, kanali, funkcije)
4. Interfejsni tipovi (interfejsi)

Konstante (`const`) u Go-u predstavljaju izraze čija je vrednost unapred poznata kompilatoru i čija se evaluacija izvršava tokom kompilacije, a ne tokom izvršavanja programa. Konstante se mogu definisati samo za osnovne tipove podataka.

Tipovi

Statički tipiziran jezik

- Go je statički tipiziran jezik što znači da se promenljivoj dodeljuje tip prilikom njene deklaracije i on se ne može menjati tokom izvršavanja programa.
- Za razliku od C-a, Go ne podržava automatsku konverziju tipova vrednosti već se konverzija vrednosti mora navesti eksplicitno, u suprotnom prijavljuje se greška prilikom kompilacije.
- Pravilo koje važi za pakete važi i za promenljive, svaka deklarisana promenljiva mora biti upotrebljena.

Tipovi

Deklaracija, definicija

Primer 2: Program koji ilustruje rad sa promenljivama

```
package main

import "fmt"

func main() {
    var a float32
    b := 5
    var c int

    c = int(a) + b
    fmt.Println(c)    // 5
}
```

Tipovi

Deklaracija, definicija

- Program prikazuje definiciju i deklaraciju različitih vrsta promenljivih.
- Tip promenljive se ne mora eksplicitno navesti, već se može zaključiti na osnovu inicijalizacije kada se promenljiva `b` uvodi.
- Iako nije definisana, promenljiva `a` ima podrazumevanu vrednost koja je za numeričke tipove 0.

```
var message string  // default value ""
var factor float32  // default value 0
var enabled bool    // false
```

Osnovni tipovi

Osnovni tipovi podataka

Osnovni tipovi koji postoje u programskom jeziku Go, podeljeni su na:

- Numeričke - celobrojne označene (`int8`, `int16`, `int32`, `int64`), celobrojne neoznačene (`uint8`, `uint16`, `uint32`, `uint64`), u pokretnom zarezu (`float32`, `float64`) i kompleksne (`complex64`, `complex128`)
- Bulovske (`bool`)
- Tekstualne (`string`)

Osobine

Operatori

Operatori koji su definisani u Go-u podeljeni su u sledeće kategorije:

- Aritmetički operatori (`+` , `-` , `*` , `/` , `%` , `++` , `--`)
- Relacioni operatori (`==` , `!=` , `>` , `<` , `>=` , `<=`)
- Logički operatori (`&&` , `||` , `!`)
- Bitski operatori (`&` , `|` , `^` , `>>` , `<<`)
- Operatori dodele (`=` , `+=` , `-=` , `*=` , `/=` , `%=` , `>>=` , `<<=` , `&=` , `|=` , `^=`)

Upotreba nad određenim tipovima podataka, uloga i arnost Go operatora, definisani su na isti način kao u jeziku C.

2.1 Složeni tipovi podataka

Složeni tipovi - nizovi i strukture

Niz

- **Niz** predstavlja sekvencu elemenata istog tipa, fiksne dužine. Elementima niza se pristupa standardnom `[indeks]` notacijom gde prvi element ima indeks 0. Ugrađena funkcija `len(niz)` koristi se za dobijanje podatka o dužini niza. Dužina niza se mora navesti prilikom deklaracije ili, ukoliko se izvršava i inicijalizacija, umesto dužine može se navesti znak `...` , a dužina će biti zaključena na osnovu incijalizacije.
- Go dozvoljava poređenje nizova iste dužine definisanih nad istim tipom podataka, relacionim operatorima `==` i `!=` . Dva niza su jednaka ako imaju jednake vrednosti na istim pozicijama.

Strukture

Strukture

- **Strukture** su složeni podaci koji grupišu nula ili više elemenata ne obavezno istih tipova. Svaki element mora biti jedinstveno imenovan i predstavlja jedno polje strukture. Definisanjem strukture definiše se novi tip podataka korišćenjem ključne reči `type` ispred same definicije strukture. Poljima strukture pristupa se pomoću znaka `.` i naziva polja. Prazne strukture `struct{}` mogu se koristiti za komunikaciju preko kanala u slučajevima kada nije bitan sadržaj poruke već samo davanje signala.
- Ugnježdavanje struktura je dozvoljeno, odnosno definisanje struktura koje sadrže druge strukture kao polja. Strukture istog tipa mogu se porediti relacionim operatorima `==` i `!=` . Dve strukture su jednake ako imaju jednake vrednosti na istim poljima.

Prenos nizova i struktura

Prenos i funkcija `new`

- Prilikom alociranja memorije za nizove i strukture umesto deklarisanja promenljivih, može se koristiti ugrađena funkcija `new` koja vraća pokazivač na alocirani niz ili strukturu.
- Nizovi se funkcijama prosleđuju kao kopija, ne preko reference kao što je slučaj u drugim jezicima poput C-a. Prenošenje niza preko reference, mora se eksplicitno naglasiti korišćenjem pokazivača. U tom slučaju, pristupanje vrednostima niza unutar funkcije postiže se korišćenjem operatora `*` odnosno notacijom `(*naziv)[indeks]`. Ukoliko se funkciji prosledi struktura preko reference, poljima se može pristupiti uobičajenom notacijom upotrebom znaka `.` i naziva polja.

Primer

Primer 3: Rad sa nizovima i strukturama

```
var a [3]int
b := [...]int{1, 2, 3}
a[0] = 1
fmt.Println(a == b) // false

type Point struct{ X, Y int }
p1 := Point{1, 2}
p2 := Point{Y:2}
p2.X = 1
fmt.Println(p1 == p2) // true

c := new([3]int)
c[0] = 1
fmt.Println(a == *c) // true
```

2.2 Referentni tipovi podataka

Referentni tipovi

Referentni tipovi

- pokazivači (kao u programskom jeziku C, pokazivač referiše na vrednost neke promenljive koja se trenutno čuva u memoriji. Operatori `&` i `*`)
- isečci — pogled na nizove, engl. *slices*, https://blog.golang.org/slices-intro
- mape (U mapama se čuvaju parovi ključ - vrednost, pri čemu je vrednost svakog ključa jedinstvena. Svi ključevi su istog tipa i sve vrednosti su istog tipa. Jedini uslov je da se tip ključa može porediti operatorom `==` , dok vrednosti mogu biti proizvoljnog tipa, uključujući i druge mape.)
- kanali
- funkcije

Funkcije

Funkcije

Funkcije u programskom jeziku Go predstavljaju referentni tip podataka. Lista parametara se navodi u formatu `naziv tip`, a ukoliko postoji više parametara istog tipa za redom dovoljno je navesti njihov tip samo jedanput. Ukoliko je lista rezultata jednočlana, nije potrebno navoditi zagrade.

Primer 4: Sintaksa za definisanje funkcije

```
func naziv(lista parametara) (lista rezultata) {
    telo funkcije
}
```

Funkcije

Povratne vrednosti i varijadičke funkcije

- Osim što mogu imati proizvoljan broj argumenata, funkcije mogu imati više povratnih vrednosti.
- Postoje i *varijadičke funkcije*, odnosno funkcije sa promenljivim brojem argumenata. Varijadičke funkcije definišu se upotrebom znaka `...` ispred tipa argumenta koji se može pojaviti nula ili više puta.

Funkcije

Funkcije

Omogućeno je definisanje funkcija unutar funkcija gde unutrašnja funkcija ima pristup svim promenljivama spoljašnje funkcije, kao i definisanje *anonimnih funkcija*, odnosno funkcija bez imena koje se izvršavaju na mestu gde su definisane. Funkcije predstavljaju tipove prvog reda i mogu se prosleđivati drugim funkcijama i dodeljivati promenljivama.

Primer 5: Rad sa funkcijama

```
func calc(x, y int) (int, int) {
    add := func() int { return x + y }
    sub := func() int { return x - y }

    return add(), sub()
}
```

Metodi

Metodi

U programskom jeziku Go ne postoji pojam klase, ali je omogućeno definisanje metoda nad korisnički definisanim tipovima. Metod predstavlja običnu funkciju koja u sebi sadrži specijalni *prijemnik* (engl. "receiver") za tip podataka nad kojim je metod definisan. Metod se definiše na isti način kao i funkcija, osim što se prijemnik navodi između ključne reči `func` i naziva metoda.

Pozivanje metoda postiže se navođenjem znaka `.` i naziva metoda nakon promenljive za koju je potrebno pozvati metod. Ukoliko metod menja podatke promenljive nad kojom je pozvan, potrebno je koristiti prijemnik sa referencom, odnosno navesti znak `*` ispred tipa prijemnika.

Metodi

Metodi

Go nema definisane nivoe vidljivosti pomoću ključnih reči kao što su `public`, `private` i `protected`. Jedini mehanizam za kontrolu vidljivosti je sledeći:

- svi identifikatori koji počinju velikom slovom, biće eksportovani van paketa,
- svi identifikatori koji počinju malim slovom, neće biti eksportovani van paketa.

Isto pravilo važi i za polja strukture. Na taj način strukture nad kojima su definisani metodi mogu da oponašaju klase i enkapsulaciju podataka.

Metodi

Primer 6: Rad sa metodima

```
type Point struct{ X, Y float64 }
func (p Point) Distance(q Point) float64 {
    return math.Sqrt((p.X-q.X)*(p.X-q.X)+(p.Y-q.Y)*(p.Y-q.Y))
}

func (p *Point) Translate(x, y float64) {
    p.X += x
    p.Y += y
}

func main() {
    p1 := Point{0,0}
    p2 := Point{1,1}
    fmt.Println(p1.Distance(p2)) // 1.41421...
    p2.Translate(5,2)
    fmt.Println(p2)              // {6 3}
}
```

2.3 Interfejsni tipovi

Interfejsi

Interfejsi

- Interfejsi se koriste kao dodatno sredstvo apstrakcije i enkapsulacije podataka. Tip interfejs definisan je kao skup potpisa metoda.
- Tip zadovoljava neki interfejs ukoliko implementira sve metode koje taj interfejs zahteva. Unutar interfejsa moguće je navoditi i nazive drugih interfejsa koji takođe moraju biti zadovoljeni.

Interfejsi

Primer 7: Rad sa interfejsima

```
type geometry interface {
    area() float64
}
```

```go
type rect struct {width, height float64}
type circle struct {radius float64}

func (r rect) area() float64 {
    return r.width * r.height
}
func (c circle) area() float64 {
    return math.Pi * c.radius * c.radius
}
func measure(g geometry) {
    fmt.Println(g, g.area())
}
```

Interfejsi

Primer 8: Rad sa interfejsima

```go
func main(){
    r := rect{width: 3, height: 4}
    c := circle{radius: 5}
    measure(r)  //{3 4} 12
    measure(c)  //{5} 78.5398
}
```

2.4 Kontrola toka

Kontrola toka

Kontrola toka

Naredbe za kontrolu toka izvršavanja su `if`, `switch`, `for` i `defer`.

Primer 9: Sintaksa naredbe `if`

```go
    if inicijalizacija; uslov1 {
        // naredbe
    } else if uslov2 {
        // naredbe
    } else {
        // naredbe
    }
```

Kontrola toka

Kontrola toka

Primer 10: Primer `if`

```go
    if num := 9; num < 0 {
        fmt.Println(num, "is negative")
    } else if num < 10 {
        fmt.Println(num, "has 1 digit")
    } else {
        fmt.Println(num, "has multiple digits")
    }
```

- `if` ne mora da ima deklaraciju i ne mora da ima `else`
- Ne postoji ternarni operator ?

Naredba switch

switch

- Naredba `switch` omogućava ispitivanje vrednosti izraza po slučajevima.
- Ukoliko je vrednost izraza jednaka nekom od slučajeva, izvršiće se deo koda naveden nakon znaka `:` , do definicije sledećeg slučaja ili kraja naredbe.
- U suprotnom izvršava se slučaj `default`, ako je naveden.
- Ukoliko se na kraju slučaja navede `fallthrough` naredba, izvršava se i kôd slučaja direktno ispod.
- Za pojedinačni slučaj moguće je navoditi više vrednosti razdvojenih zarezima.

Naredba switch

switch

Primer 11: Sintaksa naredbe `switch`

```
switch izraz {
case vrednost1:
    // naredbe
case vrednost2, vrednost3:
    // naredbe
default:
    // naredbe
}
```

Naredba switch

switch

Primer 12: Primer `switch`

```
switch i {
case 1:
    fmt.Println("one")
case 2:
    fmt.Println("two")
case 3:
    fmt.Println("three")
default:
    fmt.Println("something else")
}
```

Naredba for

Petlja for

- Jedina petlja koja postoji u programskom jeziku Go jeste `for`.
- Za izlazak iz petlje može se koristi naredba `break`, a za prelazak na sledeću iteraciju naredba `continue`.
- Petlja `for` se može koristiti zajedno sa naredbom `range` za iteriranje kroz sekvencijalne strukture kao što su nizovi, isečci i mape.

Naredba for

Primer 13: Puna sintaksa `for` petlje

```
for inicijalizacija; uslov; inkrementacija; {
    // naredbe
}

for indeks, vrednost := range sekvencijalna_struktura {
    // naredbe
}
```

Naredba for

Primer 14: Primer `for` petlje

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

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

Naredba for

Primer 15: Primer `for` petlje

```
for {
    fmt.Println("loop")
    break        // bez break ovo bi bila beskonacna petlja
}

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

Naredba for

Primer 16: Primer `for range` petlje

```
nums := []int{2, 3, 4}
sum := 0
for _, num := range nums {
    sum += num
}
fmt.Println("sum:", sum)

for i, num := range nums {
    if num == 3 {
        fmt.Println("index:", i)
    }
}
```

Naredba `defer`

Naredba `defer`

- Naredba `defer` odlaže izvršavanje navedene funkcije do trenutka završetka funkcije u kojoj se nalazi. Argumenti funkcije se određuju u trenutku navođenja naredbe `defer`, samo je izvršavanje funkcije odloženo. Naredba garantuje da će se funkcija izvršiti i u slučaju dolaska do greške.
- Najčešća upotreba naredbe `defer` je za operacije koje se obavljaju u paru, kao što je otvaranje i zatvaranje datoteka. Odmah nakon otvaranja datoteke moguće je naredbom `defer` pozvati funkciju za zatvaranje datoteke, čime se garantuje da će se datoteka zatvoriti i doprinosi čitljivijem kodu.
- open – close, connect – disconnect, lock – unlock

Naredba `defer`

Primer 17: Primer `defer`

```
func main() {
    f := createFile("/tmp/defer.txt")
    defer closeFile(f)
    writeFile(f)
    // potrebno je implementirati createFile, closeFile, writeFile
}
```

Naredba `defer`

Primer 18: Primer `defer`

```
func main() {
    i := 0
    defer fmt.Println(i)
    i++
    defer fmt.Println(i)
    return
}
```

- Šta će biti ovde odštampano?
- Ukoliko postoji više defer naredbi, one se smeštaju na stek — izvršava se najpre ona koja je poslednja postavljena

2.5 Sakupljanje smeća

Sakupljanje smeća

Paket *runtime*
Programski jezik Go ima automatsko upravljanje memorijom odnosno sakupljanje smeća (engl. *garbage collection*). Sakupljač skenira memoriju, pronalazi objekte na koje više ne postoji ni jedna referenca i nakon toga ih briše i oslobađa memoriju. Upotrebom paketa *runtime*, sakupljač se može eksplicitno pozvati korišćenjem funkcije GC().

3 Konkurentnost

3.1 Gorutine

Gorutine

Go+korutina

U programskom jeziku Go kada je u pitanju konkurentnost, osnovni pojam predstavljaju *gorutine*. Reč gorutina je nastala spajanjem reči go i reči *korutina*, koja označava funkciju koja se nezavisno izvršava. Gorutine su nešto što je karakteristično samo za jezik Go i predstavljaju niti niže kategorije. Kada se program pokrene, jedina gorutina koja postoji je glavna gorutina koja poziva funkciju `main`. Nove gorutine, kreiraju se upotrebom ključne reči `go` i navođenjem funkcije koja će se izvršavati u novoj gorutini konkurentno.

Gorutine

Primer 19: Primer gorutina

```
func main() {       //f stampa ime funkcije i u petlji brojeve 1, 2, 3
  f("direct")                    $ go run goroutines.go
  go f("goroutine")              direct : 0
  go func(msg string) {          direct : 1
        fmt.Println(msg)         direct : 2
    }("going")                   goroutine : 0
  time.Sleep(time.Second)        going
  fmt.Println("done")            goroutine : 1
}                                goroutine : 2
                                 done
```

Go planer

M:N

Gorutine se kreiraju sa stekom male veličine od nekoliko kilobajta, koji u zavisnosti od potreba može da se proširuje. Ova karakteristika omogućava kreiranje i izvršavanje velikog broja gorutina. Go poseduje sopstveni M:N planer izvršavanja gorutina koji mapira proizvoljan broj gorutina M u proizvoljan broj niti operativnog sistema N, što omogućava da se više gorutina može mapirati u jednu nit operativnog sistema.

Rantajm karakteristike

Suspenzija blokiranih niti

Rantajm sistem (engl. *runtime*) programskog jezika Go automatski suspenduje gorutine koje su postale blokirane i nastavlja njihovo izvršavanje kada postanu odblokirane. Takođe, ukoliko se više gorutina izvršava na jednoj niti operativnog sistema, ako jedna od njih postane blokirana rantajm premešta ostale gorutine na drugu nit operativnog sistema kako i one ne bi bile blokirane.

3.2 Kanali

Kanali

Operator strelica

Kanali `chan` pripadaju referentnom tipu podataka koji omogućava dvosmernu komunikaciju između dve gorutine. Tip kanala definiše tip vrednosti koju je moguće poslati kroz kanal. Operator strelice `<-` , koristi se za prosleđivanje i preuzimanje vrednosti u, odnosno iz kanala u zavisnosti sa koje strane strelice se kanal nalazi. Kanal se kreira upotrebom funkcije `make(chan tip)` u kojoj se navodi tip kanala.

Kanali

Blokiranje

U kanalu bez bafera u jednom trenutku može se nalaziti samo jedna vrednost. Ukoliko neka gorutina pokuša da preuzme vrednost iz kanala, ona postaje blokirana do trenutka dok druga gorutina ne pošalje vrednost kroz isti kanal. Slično, ukoliko vrednost iz kanala nije preuzeta, gorutina postaje blokirana ako pokuša da pošalje novu vrednost kroz kanal do trenutka dok druga gorutina ne preuzme vrednost iz tog kanala. Na ovaj način kanalima može da se postigne sinhronizacija izršavanja gorutina.

Primer

Primer 20: Upotreba kanala

```
func sum(s []int, c chan int) {
    res := 0
    for _, v := range s {
        res += v
    }
    c <- res
}
func main() {
    s := []int{1, 2, 3, 4, 5, 6}
    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c
    fmt.Println(x, y, x+y)  // 15 6 21
}             // redosled ispisa brojeva 15 i 6 varira
```

Kanali

Kanali

- Pošiljalac može zatvoriti kanal upotrebom funkcije `close(kanal)`. Za razliku od datoteka kanale nije neophodno zatvoriti.
- Zatvaranje kanala se koristi kako bi primalac znao da se nove vrednosti više neće slati. Slanje kroz zatvoreni kanal dovodi do greške.
- Moguće je definisati kanale sa baferom kroz koji se mogu poslati više od jedne vrednosti bez blokiranja prilikom slanja.
- Veličina bafera definiše se prilikom kreiranja kanala u funkciji `make`. Podrazumevana vrednost bafera je 1.
- Funkcija `cap(kanal)` koristi se za dobijanje veličine bafera kanala.

Kanali

Kanali

- Osim dvosmernih mogu se definisati i jednosmerni kanali upotrebom operatora strelice.
- Smer kanala označen je pozicijom operatora strelice u odnosu na ključnu reč `chan`.
- Konverzija dvosmernog kanala u jednosmerni je dozvoljena dok suprotna situacija nije.

Primer

Primer 21: Upotreba jednosmernog kanala i kanala sa baferom

```
func fibonacci(n int, c chan<- int) {
    x, y := 0, 1 // svaku izracunatu vrednost
    for i := 0; i < n; i++ {  // salje kroz kanal
        c <- x // U kanal se upisuje vrednost x
        x, y = y, x+y
    }
    close(c)
}
func main() {
    c := make(chan int, 10)
    go fibonacci(cap(c), c) // konv. dvosmernog kanala u jednosmerni
    for i := range c { // iz kanala se redom ucitavaju sracunate vrednosti
        fmt.Println(i) // 0 1 1 2 3 5 8 13 21 34
    }
}
```

3.3 Sinhronizacija

Sinhronizacija

- Programski jezik Go pored kanala, podržava i druge vidove sinhronizacije
- Paket `sync`
- U okviru paketa `sync` nalaze se strukture za sinhronizaciju kao što su `Mutex` (semantika katanca), `WaitGroup` (omogućava da se kreirane gorutine sačekaju da završe posao), `atomic` (omogućava atomičko izvršavanje neke funkcije)
- U programskom jeziku Go postoji specijalna naredba `select` koja funkcioniše kao naredba `switch` i upotrebljava se samo za kanale. Kao slučajevi se navode naredbe slanja ili preuzimanja, u odnosno iz kanala. Naredba je blokirana do trenutka kada neki od slučajeva može da se izvrši. Ukoliko je u jednom trenutku moguće izvršiti više slučajeva, jedan se bira na slučajan način.

4 Literatura

Literatura

- Slajdovi su sastavljeni na osnovu materijala iz master rada Miloša Mitrovića "Konkurentno programiranje u programskom jeziku Go" (mentor Milena Vujošević Janičić), koji je ujedno i preporučena literatura na temu konkurentnosti u programskom jeziku Go
- https://gobyexample.com/
- https://golang.org/