Uvod u programski jezik Go

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Luka Marković, Tamara Radovanović, Rade Aleksić, Milan Pužić
luka.markovic.d@gmail.com, radovanovic.tamara.t@gmail.com,
aleksic0rade@gmail.com, milanpuzic@gmail.com

5. april 2019.

Sažetak

Ovaj seminarski rad je posvećen programskom jeziku Go. On spada u mlade jezike sa jednostavnim i čitljivim kodom. Zbog svojih karakteristika postao je popularan za svega nekoliko godina. U radu su prikazane njegove osnove, sintaksa i neki primeri koda. Prvi deo se bavi istorijom i razvojem, kao i radnim okruženjem. Nakon toga sledi opis osnovnih delova programa pisanih u Go-u. Na samom kraju predstavljen je način na koji se u ovom jeziku ostvaruje konkurentno programiranje.

Sadržaj

1 Uvod

Go je programski jezik otvorenog koda koji olakšava izradu jednostavnog, pouzdanog i efikasnog softvera [4]. Go je imperativni, statički tipizirani, kompajlirani jezik. Zbog svojih karakteristika za njega se kaže da je C 21. veka [1]. On nije objektno orijentisan, ali preuzima neke od koncepata iz objektno orijentisanih jezika kao što su metodi koji se dodeljuju korisnički definisanim tipovima i interfejsi. Go se često koristi za izradu serverskih aplikacija, zbog svoje konkurentnosti.

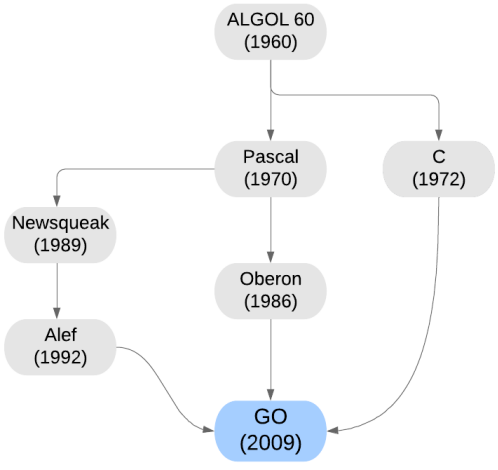
2 Nastanak i razvoj

U ovom odeljku su opisani motivi za nastanak jezika Go, kao i uticaj drugih jezika na njegov razvoj. Pored toga prikazane su njegove osnovne karakteristike i namene.

2.1 Istorija

Programski jezik Go je nastao 2007. godine i kao takav spada u mlade programske jezike. Javno je predstavljen 2009. godine kao projekat otvorenog koda nastao unutar kompanije Google. Jezik su osmislili Robert Griesemer, Rob Pike i Ken Thompson, a njihov osnovni cilj bio je da naprave novi jezik opšte namene. Go je nastao kao pokušaj da se ukombinuju lak način pisanja i čitanja interpretiranih jezika i efikasnost i sigurnost jezika koji se kompajliraju.

Go se razvijao pod uticajem drugih jezika, preuzimajući njihove ideje i izbegavajući karakteristike koje dovode do komplikovanog koda. Može se reći da pripada familiji programskog jezika C, iako je koncept konkurentnosti preuzeo iz Limbo i Newsqueak jezika. Mesto Go jezika u razvojnom stablu se može videti na slici 1. Slična sintaksa, ali znatno pojednostavljena i čistija u odnosu na jezik C, omogućila je velikom broju programera da se lako upoznaju sa ovim jezikom [2].



Slika 1: Razvojno stablo

2.2 Karakteristike i poređenje sa drugim jezicima

Programeri koji su dizajnirali Go su želeli jednostavan jezik. Iz tog razloga, napravljen je jezik koji nije objektno orijentisan. U Go-u ne postoje klase, koncept nasleđivanja kao ni konstruktori, izuzeci ili anotacije. Time se Go razlikuje od mnogih jezika koji su dodavanjem velikog broja funkcionalnosti postali suviše složeni i često komplikovani za rad, kao što je npr. C++.

Jedna od prednosti koje donosi ova jednostavnost je brzina kompilacije. Go se brzo kompilira, direktno u mašinski jezik i proizvodi jedan izvršni fajl, bez spoljnih dinamičkih biblioteka i bez potrebe za virtuelnom mašinom.

Druga prednost jednostavne sintakse je lakša čitljivost koda. Ovome doprinosi i postojanje jedinstvenog standarda za formatiranje koda pisanog u Go-u.

Kako bi olakšali programiranje, jezik ima automatsko upravljanje memorijom, odnosno sakupljanje otpada. U odnosu na druge jezike, kao što je Java, Go ima efikasniji sakupljač otpada sa izuzetno niskim kašnjenjem.

Iako statički tipiziran, preuzeo je i karakteristike dinamički tipiziranih jezika kao što su Python i Ruby. Zato on pruža efikasnost statički tipiziranih jezika, ali i lakoću programiranja koju obično nude dinamički tipizirani jezici. Takođe, nema implicitnih konverzija tipova. Zahvaljujući ovim činjenicama, Go ima bolju bezbednost tipova od Jave, C-a i C++-a.

Kako je nastao u eri multiprogramiranja, on omogućava efikasnu konkurentnost koja je ugrađena u sam jezik i njegov je integralni deo. Ovo ga razlikuje od većine drugih jezika za sistemsko programiranje, kao što su C ili C++, koji uglavnom nemaju ugrađenu podršku za konkurentno programiranje. Go podržava gorutine umesto niti (detaljnije objašnjeno u odeljku 6). One zauzimaju oko 2 KB memorije, što je mnogo manje nego 1 MB koliko zauzima svaka nit u Javi. Sve ovo čini Go jezikom visokih performansi [7].

Go je prvenstveno osmišljen kao jezik za pisanje sistemskog softvera, pa je u toj oblasti i pronašao najveću primenu. Osim toga koristi se i za:

- programiranje distribuiranih sistema (Ethereum, ipfs)
- programiranje DevOps alata (Docker, Prometheus)
- pisanje alata za komandnu liniju (fzf, rclone)
- baze podataka (influxdb)
- razvoj veb aplikacija (gin, echo, iris)
- razvoj video igara
- mašinsko učenje
- mnoge druge oblasti

2.3 Instalacija i korisničko okruženje

Da bi se programiralo koristeći programski jezik Go, sve što je potrebno su tekst editor i kompajler za Go. Može se koristiti bilo koji editor,s tim što je pogodnije izabrati one koji podržavaju sintaksu ovog jezika. Za većinu editora moguće je instalirati paket koji omogućava lakše pisanje koda. Pored ovoga, postoje razvojna okruženja specifično namenjena za ovaj jezik. Među njima je najpoznatije okruženje GoLand [5].

Bilo u slučaju Windows-a, Linux-a ili macOs-a, poslednja verzija kompajlera za Go kao i uputstvo za instalaciju može se naći na zvaničnoj strani

jezika [4]. Ekstenzija izvornog koda programa napisanog u Go jeziku je .go, a nakon njegove kompilacije dobija se izvršni kod sa ekstenzijom .out na Unix sistemima i .exe na Windows operativnim sistemima.

3 Osnovna sintaksa

U daljem tekstu je prvo prikazana minimalna struktura za svaki program pisan u jeziku Go, a potom se dalje razmatraju osnovni elementi jezika. Program u osnovi sadrži sledeće delove:

- Deklaraciju paketa
- Uključivanje drugih paketa u program
- Funkcije
- Promenljive
- Naredbe za kontrolu toka i naredbe grananja
- Komentare

Primer 3.1 *Primer programa u listingu 1 ispisuje poruku "Zdravo, svete!" na standardnom izlazu.*

```
1 package main
2 import "fmt"
3 func main() {
4     fmt.Println("Zdravo. svete!")
5 }
```

Listing 1: Zdravo svete

Prva linija programa definiše ime paketa u kome se program nalazi. Paket `main` je početna tačka za izvršavanje programa. Svaki paket ima pridruženu putanju i ime. Sledeća linija je pretprocesorska komanda koja govori Go kompajleru da uveze fajlove koji se nalaze u paketu `fmt`. Funkcija `main()` je funkcija od koje počinje izvršavanje programa. Kada operativni sistem pokrene program, on prvo poziva ovu funkciju. Tekst ograničen karakterima `/*...*/` označava komentare. Komanda `Fmt.Println("Zdravo, svete")` poziva funkciju `Println` iz paketa `fmt` koja ispisuje na standardni izlaz poruku "Zdravo, svete".

Naredbe u programskom jeziku Go se razdvajaju novim redom. Nije obavezno navoditi karakter `;` kao na primer u programskim jezicima C i Java. Moguće je napisati više naredbi u istom redu, ali se tada one moraju razdvajati karakterom `;`.

3.1 Komentari i identifikatori

Komentari u Go-u mogu biti jednolinijski ili višelinijski 2. Višelinijski komentari su ograničeni između sekvenci karaktera `/*` i `*/`. Jednolinijski komentari počinju sekvencom karaktera `//`.

```
1 /* Ovo je komentar
2 koji se prostire
3 na vise linija */
4
5 // Ovo je jednolinijski komentar
```

Listing 2: Komentari

Identifikatori su imena koja jednoznačno određuju promenljive, funkcije i sve ono što korisnik definiše. Imena identifikatora počinju velikim ili malim slovom ili karakterom _, a zatim su opciono praćeni kombinacijom slova, brojeva i karaktera _. Izvorne datoteke sa kodom su uvek kodirane sa UTF-8. Validna imena identifikatora u jeziku Go su:

```
imePromenljive      DrugoIme_promenljive      Ћирилица30
```

Programski jezik Go razlikuje velika i mala slova (eng. *case-sensitive*). Iz tog razloga su identifikator `foobar` i identifikator `FooBar` dva različita identifikatora.

Tabela 1 prikazuje ključne reči u Go-u. To su reči koje su rezervisane i ne mogu se upotrebljavati ni za imena promenljivih i konstanti, niti za bilo koji drugi identifikator.

Tabela 1: Rezervisane reči u jeziku Go

break	default	func	interface	select
case	defer	Go	map	Struct
char	else	GoTo	package	Switch
const	fallthrough	if	range	Type
continue	for	import	return	Var

3.2 Promenljive i konstante - deklaracija i inicijalizacija

Pomoću ključne reči `var` se kreira promenljia određenog tipa, dodeljuje joj se neko ime i inicijalizuje vrednost. Svaka deklaracija promenljive ima sledeću formu: `var ime tip = izraz`. Moguće je deklarisati i inicijalizovati više promenljivih u jednoj liniji 3.

Jedna od vrednosti `tip` ili `izraz` može biti izostavljena, ali nikako obe. Ako se izostavi `tip`, tip se dodeljuje na osnovu vrednosti izraza. Ako se izostavi `izraz`, onda se vrednost promenljive inicijalizuje na "nula" za određen tip. Za numeričke vrednosti to je `0`, za bulovske vrednosti to je `false`, za string `""`, a za referencne vrednosti to je `nil`.

"Nula vrednost" mehanizam obezbeđuje da svaka promenljiva u svakom trenutku ima validnu vrednost određenog tipa. Dakle, u programskom jeziku Go ne postoje neinicijalizovane promenljive.

```
1 var x, y, z Int // Sve promenljive su tipa int i imaju vrednost 0
2 var x, y, z = 2, true, 0 // Promenljive x i z su tipa int i imaju
      vrednost 2 i 0, a y je tipa bool i ima vrednost true
3 const pi = 3.14 // Deklaracija pomocu kljucne reci const
```

Listing 3: Inicijalizacija i deklaracija više promenljivih

Unutar funkcija je moguće koristiti još jedan način deklarisanja. Kratka deklaracija promenljivih ima oblik: `Ime := izraz`.

Deklaracija je moguća i pomoću ključne reči `const`, što se vidi u listingu 3. U ovom slučaju sintaksa je slična onoj kod koje se koristi ključna reč `var`, ali definiše imenovanu vrednost koja mora da bude konstanta. Vrednost mora biti poznata u toku kompajliranja programa. Tip konstantne promenljive mora biti osnovni tip, odnosno numerička vrednost, bulovska vrednost ili string.

4 Tipovi i strukture podataka

Go je statički tipiziran jezik što znači da se promenljivoj dodeljuje tip prilikom njene deklaracije i on se ne može menjati tokom izvršavanja programa. Tip promenljive se ne mora eksplicitno navesti, već se može zaključiti na osnovu dodele operatorom `:=` kada se promenljiva uvodi.

Tipovi podataka koji su definisani u programskom jeziku Go mogu se klasifikovati u četiri kategorije [2]:

1. Osnovni tipovi
2. Složeni tipovi
3. Referentni tipovi
4. Interfejsni tipovi

4.1 Osnovni tipovi

U osnovne tipove u programskom jeziku Go spadaju numerički, stringovski, i bulovski tipovi. Numerički tipovi uključuju nekoliko tipova celih brojava (eng. *integer*), brojeva sa pokretnim zarezom (eng. *floating point*) i kompleksnih brojeva.

Celi brojevi mogu da budu označeni i neoznačeni. Postoje 4 vrste celih brojeva u programskom jeziku Go i oni su klasifikovani po veličini, preciznije memoriji koja je potrebna za njihovo čuvanje. To su reprezentacije celih brojeva od 8, 16, 32 i 64 bita i u Go-u se označavaju respektivno sa: `int8`, `int16` i `int32`, `int64` za označene brojeve i sa `uint8`, `uint16`, `uint32` i `uint64` za neoznačene brojeve.

Cele brojevi se označavaju i samo sa `int` ili `uint` i ovi tipovi su predstavljeni u memoriji sa 32 ili 64 bita zavisno od hardvera. Operacije koje se mogu izvesti nad celim brojevima su prikazane u tabeli 2.

Tabela 2: Aritmetičke i logičke operacije nad celim brojevima

+	Sabiranje	*	Množenje
-	Oduzimanje	/	Deljenje
%	Ostatak pri deljenju	++	Inkrementacija
–	Dekrementacija	==	Ispitivanje jednakosti
!=	Nejednakost	>,>=,<,<=	Relacije veće, manje...
&&	Logičko I	||	Logičko ili
!	Logička negacija	=	Operator dodele

U programskom jeziku Go postoje dva tipa brojeva sa pokretnim zarezom i označavaju se sa `float32` i `float64`. Njihova fizička reprezentacija kao i aritmetičke operacije određene su IEEE 754 standardom čiju implementaciju podržava većina modernih procesora.

Postoje dve bulovske vrednosti, to su `true` i `false`. Promenljive ili konstante koje čuvaju bulovsku vrednost su tipa `bool`.

Stringovi su nepromenljiva sekvenca bajtova. Oni mogu da čuvaju proizvoljne podatke, ali najčešće je to tekst razumljiv čoveku. Tekst stringova je predstavljen kao UTF-8 enkodirana sekvenca. Stringovi su nepromenljivi (eng. *immutable*), što znači da sekvenca bitova nikada ne može biti promenjena. Na primer, ako je string `s` deklarisan sa `s := "String"`, pokušaj da se promeni prvo slovo, `s[0] = 'R'`, bi javljao grešku kompajlera. Ako se uradi konkatenacija dva stringa operatorom `+`, `s = s +`

`"mutirani"`, ne bi se promenio prvobitni string `s`, već bi se napravio novi string `"String mutirani"` i dodelio promenljivoj `s` [3].

4.2 Složeni tipovi podataka

Kako bi omogućio čitljiviji kod Go podržava složene tipove podataka. U njih spadaju nizovi i strukture koji su prikazani u nastavku.

Niz je sekvenca, fiksne dužine, od 0 ili više elemenata istog tipa. Pojedinačnom elementu niza može se pristupiti preko njegovog indeksa. Indeksiranje u Go-u počinje od 0, što znači da prvi element u nizu ima indeks 0, drugi 1...

```
var a [3]int // Deklaracija niza od tri cela broja
fmt.Println(a[0]) // Ispisuje prvi element na standardni izlaz
```

Listing 4: Deklaracija nizova

Podrazumevano, elementi niza su inicijalno postavljeni na 0 ako se drugačije ne naglasi. Niz je moguće inicijalizovati prilikom deklaracije 5.

```
var niz [3]int = [3]int{1, 12, 3} // Prvi element je 1, drugi 12,
    treci 3
```

Listing 5: Inicijalizacija niza prilikom deklaracije

Veličina niza je deo njegovog tipa, dakle niz `[3]Int` i niz `[4]Int` su različiti tipovi. Nije moguće dodeliti `[4]Int` na `[3]Int` 6. Veličina mora biti konstantan izraz, čija vrednost može da se izračuna kada se program kompajlira.

```
q := [3]int{1, 2, 3}
q = [4]int{1, 2, 3, 4} // Greska u kompilaciji
```

Listing 6: Primer greške u radu sa nizovima

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.

Struktura je tip podatka koja grupiše zajedno nula ili više različitih podataka. Svaki podatak ima svoje ime i svoju vrednost. Definisanjem strukture definiše se novi tip pomoću ključne reči `type`, zatim se navodi ime novog tipa, pa ključna reč `struct` i onda definicija strukture unutar zagrada `{}`.

```
type Automobil struct {
  proizvodjac String,
  marka String,
  snaga Int
}

var golf Automobil;
```

Listing 7: Definicija strukture

Pojedinačnim poljima se pristupa pomoću tačka notacije. U primeru 7, da bi se saznala snaga automobila `golf`, mora se uraditi sledeće: `golf.snaga`. Ako je moguće uporediti polja strukture određenog tipa, onda je moguće uporediti i same strukture tog tipa operatorima `==` i `!=`.

4.3 Referentni tipovi podataka

U ovom odeljku će biti reči o referentnim tipovima podataka. U njih spadaju pokazivači, isečci, kanali, mape, kao i funkcije koje će biti detaljnije objašnjene u odeljku 5.3.

Pokazivači su vrsta podataka koji čuvaju memorijsku adresu neke promenljive. Dakle, pokazivač čuva lokaciju u memoriji na kojoj je sačuvan neki podatak imenovane promenljive. Pomoću pokazivača je moguće pristupiti i menjati vrednost promenljive bez znanja imena same promenljive. Ako je promenljiva deklarisana sa `var x int` onda je operatorom `&` moguće dobiti adresu promenljive `x`.

```
x := 1
p := &x // Pokazivac na promenljivu x
```

Listing 8: Primer pokazivača

Vrednost na koju pokazuje neki pokazivač mogu se dobiti operatorom `*`. U primeru 8, vrednost na koju pokazuje `p` dobija se sa `*p`. Takva vrednost može se izmeniti naredbom `*p = 2`. "Nula" vrednost za pokazivač na bilo koji tip je `nil`. Pokazivači su uporedivi, dva pokazivača su jednaka ako pokazuju na istu memorijsku adresu ili ako su oba `nil`. Oni se mogu kreirati i bez promenljive 9. Izraz `new(E)` kreira neimenovanu promenljivu tipa `E`, inicijalizuje je na "nula" vrednost tipa `E` i vraća adresu te promenljive.

```
p := new(int)  // p je tipa *int i pokazuje na neku neimenovanu
    promenljivu
```

Listing 9: Primer kreiranja pokazivača

Isečak (eng. *slice*) čini sekvencu elemenata promenljive dužine, čiji elementi imaju isti tip. Isečak se označava sa `[]T`, gde elementi imaju tip `T`. Podseća na niz, samo bez unapred određene dužine. Isečak ima tri komponente: pokazivač, dužinu i kapacitet. Pokazivač pokazuje na prvi element u isečku. Dužina je broj elemenata u isečku i ona ne može da bude veća od kapaciteta. Dok je kapacitet alociran broj elemenata za isečak, odnosno broj elemenata za koji je rezervisan prostor u memoriji.

Kanali (eng. *channels*) omogućavaju dvosmernu komunikaciju između dve gorutine. Oni će biti detaljnije opisani u odeljku 6.

U programskom jeziku Go, mapa je referentni tip podataka koja referiše na heš tabelu. Heš tabela je neuređena kolekcija sačinjena od parova ključ-vrednost. Mapa se označava sa `map[K]V`. `K` je tip ključa i on mora biti uporediv operatorom `==`, a `V` je tip vrednosti. Ključ mora da bude jedinstven za svaku vrednost u tabeli. Mapa se može kreirati inicijalizacijom vrednosti ili korišćenjem funkcije `make` 10.

```
godine := map[string]int{ // Kljuc tipa string, a vrednost int
"tamara": 24,
"milan": 23,
}
// Kod iznad je ekvivalentan sa:
godine := make(map[string]int)
godine["tamara"] = 24
godine["milan"] = 23
```

Listing 10: Kreiranje mape

4.4 Interfejsni tipovi podataka

Konkretni tipovi predstavljaju egzaktne reprezentacije njihovih vrednosti i daju operacije koje se mogu izvršiti nad tim tipom, na primer aritmetičke operacije za brojeve ili indeksiranje za nizove i isečke. Konkretnom tipu takođe mogu da se proslede dodatna ponašanja kroz metode. Kada postoji vrednost nekog konkretnog tipa, tačno se zna šta je ona i šta može sa njom da se uradi.

Pored konkretnih tipova u Go-u postoje i interfejsni tipovi podataka ili drugačije interfejsi 11. Oni izražavaju apstrakciju ili generalizaciju ponašanja nekog drugog tipa. Pomoću generalizacije, intefejsi dozvoljavaju da se pišu funkcije koje su fleksibilnije i bolje adaptirane zato što ne ulaze u detalje određene implementacije.

Interfejs je apstraktan tip. On ne pokazuje reprezentaciju interne strukture svojih vrednosti ili osnovene operacije koje one podržavaju. Interfejs otkriva samo neke od svojih metoda. Kada postoji neka vrednost interfejsnog tipa, ne zna se šta je ona, zna se samo šta ona može da uradi, tačnije koje ponašanje obezbeđuje preko svojih metoda. Jedan interfejs određuje skup metoda koji konkretan tip mora da implementira da bi bio razmatran kao tip tog interfejsa.

```
type ime interface {
   metod1 [return_type]
   metod2 [return_type]
   ...
}
```

Listing 11: Primer interfejsa

Moguće je da se jedan interfejs definiše kao kombinacija više drugih interfejsa, nešto kao nasleđivanje u Javi. U primeru 12 su definisani interfejs `A`, kao kombinacija interfejsa `B` i `C`. Dakle, neki korisnički tip može biti razmatran kao tip interfejsa `A`, ako može i kao tip interfejsa `B` i `C`.

```
type A interface {
   B
   C
}
```

Listing 12: Primer interfejsa definisanog preko drugih interfejsa

5 Kontrola toka

Naredbe kontrole toka se koriste za menjanje toka izvršavanja programa. To se postiže grananjem, petljama ili drugim načinima izvršavanja koda na osnovu zadovoljenih uslova.

5.1 Naredbe grananja

U programskom jeziku Go postoje dve naredbe grananja. Prva je `if`, odnosno `if-else`, a druga `switch` naredba.

Naredbom `if-else` se proverava uslov i u zavisnosti od istinite vrednosti tog uslova izvršava se jedan blok naredbi. Ova naredba se može napisati sa izostavljanjem `else` dela. Ovo je prikazano u listinigu 13.

```
if uslov1 {
//Izvrsava ovaj blok naredbi ako je uslov1 tacan
} else if uslov2 {
//Izvrsava ako je uslov2 tacan i uslov1 nije tacan
}
else {
//Izvrsava u slucaju da uslov1 nije tacan i uslov2 nije tacan
}
```

Listing 13: Primer `if` naredbe

Naredba `switch`, koja je prikazana u listingu 14, se koristi kako bi se izbeglo gomilanje `if-else` grananja. Ona daje lepšu i čistiju sintaksu za ispitivanje većeg broja slučajeva.

```
switch broj {
case 1: // Prvi slucaj
case 2: // Drugi slucaj
case 3: // Treci slucaj
case 4: // Cetvrti slucaj
default: // Podrazumevani slucaj
}
```

Listing 14: Primer `switch` naredbe

5.2 Petlje

Jedina petlja koja postoji u programskom jeziku Go je `for`. Naredni listing 15 pokazuje kako se ona definiše.

```
for inicijalizacija; uslov; inkrementacija {
//Blok naredbi
}
```

Listing 15: Primer `for` petlje

Za izlazak iz petlje može da se koristi naredba `break`, a za prelazak na drugu iteraciju naredba `continue`. Postoji još jedan oblik naredbe `for`, koja se koristi u kombinaciji sa naredbom `range` za prolazak kroz iterativne sekvencijalne strukture kao što su niz, isečak ili mapa. Ovo je prikazano u listingu 16.

```
for indeks, vrednost := range struktura {
//Blok naredbi
}
```

Listing 16: Primer `for-range` petlje

5.3 Funkcije

Funkcije u programskom jeziku Go predstavljaju referentni tip podataka. Definicija funkcija počinje sa ključnom reči `func`, zatim slede ime funkcije, lista parametara i lista povratnih vrednosti. Deklaracija funkcije je prikazana u narednom primeru 17.

```
func ime_funkcije(lista_parametara) (tipovi_povratnih_vrednosti)
{
//Blok naredbi
}
```

Listing 17: Deklaracija fukcije

Lista parametara predstavlja listu tipova i imena parametara i oni se prosleđuju prilikom pozivanja funkcije. Povratne vrednosti predstavljaju listu tipova povratnih vrednosti. Ako funkcija vraća jednu neimenovanu vrednost ili ne vraća rezultat uopšte, onda lista povratnih vrednosti može biti izostavljena.

```
func zbir(x, y Int) Int {
   return x+y
}
```

Listing 18: Primer funkcije

Poziv funkcije bi izgledao ovako:

```
x:=zbir(5, 2) // Dakle x je 7
```

Kao što je prikazano u primeru 18, parametari istog tipa se mogu grupisati, tako da jedan tip ne mora da se piše više puta. Tip funkcije je ono što se naziva potpis. Dve funkcije imaju isti tip ili imaju isti potpis ako imaju istu listu tipova parametara i istu listu tipova povratnih vrednosti. Imena parametara ne utiču na tip funkcije, odnosno na potpis.

Velika razlika u odnosu na ostale popularne programske jezike višeg nivoa, kao što su Java, C ili C++ je što kod programskog jezika Go može postojati više povratnih vrednosti za jednu funkciju 19. Ovo je koristan koncept kada je potrebno upravljati izuzecima i greškama u programu. Prva vrednost može biti ono što je cilj izračunavanja naše funkcije, a druga vrednost - vrsta greške (ako je do nje došlo).

```
func ime_funcije(parametri) (string, string) {
  return "Prvi string", "Drug student"
}
```

Listing 19: Primer funkcije sa više povratnih vrednosti

Metod je pojam koji je uveden u objektno orijentisanom programiranju i predstavlja funkciju koja je dodeljena korisnički definisanom tipu podataka (najčešće klasama). Dakle metoda je funkcija koja je vezana za određeni tip podatka. Iako u Go-u ne postoje klase, programerima je omogućeno da definišu metode za neki tip podatka. U primeru 20 je prikazana razlika u načinu definisanja metoda i funkcija.

```
package geometrija

import "math"

type Tacka struct{ X, Y float64 }

// Tradicionalna funkcija
func Distanca(p, q Tacka) float64 {
return math.Hypot(q.X - p.X, q.Y - p.Y)
}
// Metod za tip Tacka
func (p Tacka) Distanca(q Tacka) float64 {
return math.Hypot(q.X - p.X, q.Y - p.Y)
}
p := Tacka{1, 2}
q := Tacka{4, 6}
fmt.Println(Distanca(p, q)) // "5", racunanje pomocu funkcije
fmt.Println(p.Distanca(q)) // "5", poziv preko metoda
```

Listing 20: Primer metoda

6 Konkurentno programiranje

Pojam konkurentnog programiranja se odnosi na mogućnost da se delovi programa izvršavaju nezavisno jedni od drugih. To omogućava njihovo paralelno izvršavanje bez uticaja na rezultat.

U programskom jeziku Go, konkurentno programiranje se ostvaruje pomoću gorutina. Gorutine su slične nitima, ali za razliku od njih, gorutinama upravlja Go, a ne operativni sistem. Gorutine se jednostavno kreiraju pomoću ključne reči `go` nakon koje sledi funkcija koja se izvršava konkurentno. Veći broj gorutina (M) se može preslikati u proizvoljan broj niti (N) operativnog sistema što predstavlja M:N model niti [6].

Za dvosmernu komunikaciju između niti se koriste kanali koji predstavljaju referentni tip podataka. Pomoću operatora `<-` vrednosti se smeštaju u kanal ili preuzimaju iz njega, zavisno od toga sa koje strane strelice se kanal nalazi. Kanali se kreiraju korišćenjem funkcije `make` u kojoj se navodi tip kanala. Primer kreiranja niti i korišćenja kanala prikazan je u listingu 21.

```
1 poruka1 := make(chan string) //Kanal tipa string
2 go func() {poruka1 <- "go_kanal"}() //Smestanje vrednosti u kanal
3 poruka2 := <-poruka1 //Preuzimanje vrednosti iz kanala
```

Listing 21: Kreiranje kanala

7 Zaključak

U radu su predstavljene osnovne funkcionalnosti i mogućnosti jezika Go, koje predstavljaju osnovu za dalja istraživanja. Go pruža visoke performanse nalik jezicima C i C++, izuzetno efikasno upravljanje konkurentnošću nalik jeziku Java i jednostavan je za kodiranje kao jezici Python ili Perl. Ima primenu u raznim oblastima kao što su grafika, mobilne aplikacije, mašinsko učenje i još mnoge druge.

Literatura

[1] I. Balbaert. *The Way to Go: Introduction to the Go Programming Language*. iUniverse, 2012.

[2] B. W. Donovan, A. A. A. i Kernighan. *The Go Programming Language*. Addison-Wesley Professional, 2015.

[3] C. Doxseyt. *An Introduction to Programming in Go*. CreateSpace Independent Publishing Platform, 2012.

[4] Google. Golang, 2009. on-line at: https://golang.org/doc/install.

[5] JetBrains s.r.o. GoLand, 2000-2019. on-line at: https://www.jetbrains.com/go/.

[6] M. Marić. *Operativni sistemi*. Univerzitet u Beogradu - Matematički fakultet, 2015.

[7] S. Nanz and C. A. Furia. A Comparative Study of Programming Languages in Rosetta Code. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 778–788, May 2015. on-line at: https://ieeexplore.ieee.org/abstract/document/7194625.