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 rad je posvećen programskom jeziku Go. On spada u mlade jezike sa jednostavnim i čitljivim kodom. U ovom radu biće prikazane osnove jezika Go, njegova sintaksa i neki primeri koda. U prvom delu ćemo se osvrnuti na razvoj jezika i radno okruženje. Nakon toga biće reči o osnovnim delovima programa pisanih u Go jeziku. Za sam kraj biće predstavljeno na koji način Go ostvaruje konkurentno programiranje.

Sadržaj

1 Uvod

Go je programski jezik opšte namene, razvijen u kompaniji Google. Ideja je bila osmišljavanje novog, jednostavnog i moćnog jezika. Go je imperativni, statički tipizirani, kompajlirani jezik. Zbog svojih karakteristika za njega se kaže da je C 21 veka [**?**]. 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 je dobar za izradu serverskih apikacija, zbog svoje konurentosti. Zbog svih ovih karakteristika postao je popularan u kratkom vremenskom periodu.

2 Nastanak i razvoj

U ovoj glavi će biti opisani razlozi za nastanak jezika Go, koji su jezici uticali na njegov razvoj, kao i neke 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 je nastao pod uticajem raznih 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 **??**. Slična sintaksa, ali znatno pojednostavljena i čistija u odnosu na jezik C, omogućila je velikom broju programera da se lako upoznaju sa ovim jezikom. [**?**]

2.2 Karakteristike

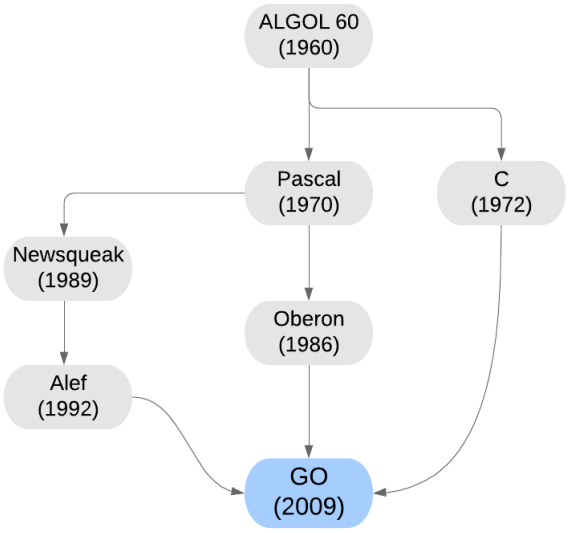
Programeri koji su dizajnirali Go su želeli jednostavan jezik. Iz tog razloga, napravljen je jezik koji nije objektno orjentisan. U Go-u ne postoje klase, koncept nasleđivanja kao ni konstruktori, izuzeci ili anotacije. To Go čini drugačijim od drugih jezika.

Kako bi olakšali programiranje, Go ima automatsko upravljanje memorijom, odnosno sakupljnje otpada. Iako statički tipiziran, preuzeo je i karakteristike dinamički tipiziranih jezika kao što su Python i Ruby. Kako je nastao u eri multiprogramiranja, Go omogućava efikasnu konkurentnost koja je ugrađena u sam jezik. Go podržava gorutine umesto niti. One zazuimaju oko 2KB memorije, što je mnogo manje nego 1MB koliko zauzima svaka nit u Javi. Sve ovo čini Go brzim i moćnim jezikom [**?**].

2.3 Instalacija i korisničko okruženje

Kako bismo programirali koristeći programski jezik Go, sve što nam je potrebno su tekst editor i kompajler za Go.

Može se koristiti bilo koji tekst editor, s tim što bi bilo poželjno da podržava sintaksu programskog jezika Go. Za većinu text editora moguće je instalirati paket koji omogućava lakše pisanje koda.



Slika 1: Razvojno stablo

Bilo da koristite Windows, Linux ili macOS, poslednju verziju kompajlera za Go kao i uputstvo za instalaciju možete naći na zvaničnoj strani Go jezika [**?**].

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

Programi pisani u jeziku Go se takođe mogu pisati u razvojnom okruženju. Postoji više razvojnih okruženja među kojima je najpoznatije GoLand [**?**].

3 Osnovna sintaksa

Pre nego što krenemo dalje da razmatramo osnovne elemente programskog jezika Go, pogledajmo prvo minimalnu strukturu za svaki program pisan u njemu.

Program pisan u Go-u u osnovi sadrži sledeće delove:

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

Primer 3.1 *Pogledajmo jednostavan primer* **??** *koji na standardnom izlazu ispisuje poruku "Zdravo svete!"*

```
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 u programskom jeziku Go. 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 jednolijski ili višelinijski. 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 Go programskom jeziku:

```
1 imePromenljive     DrugoIme_promenljive     Ћирилица30
```

Listing 3: Imenovanje identifikatora

Programski jezik Go razlikuje velika i mala slova (eng. case-sensitive), što znači da su identifikator `foobar` i identifikator `FooBar` dva različita identifikatora.

Sledeća tabela **??** prikazuje ključne reči u programskom jeziku Go. 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` kreiramo promenljivu određenog tipa, dodeljujemo joj neko ime i inicijalizujemo vrednost. Svaka deklaracija promenljive ima sledeću formu: `var ime tip = izraz`

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 `""`, 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.

Moguće je deklarisati i inicijalizovati više promenljivih u jednoj liniji.

```
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
```

Listing 4: Inicijalizacija i deklaracija više promenljivih

Unutar funkcija je moguće koristiti takozvanu kratku deklaraciju promenljivih koja ima oblik: `Ime := izraz`

Deklaracija je moguća i pomoću ključne reči `const`. U ovom slučaju sintaksa je slična onoj kod koje se koristi ključna reč `var`, ali definiše imenovanu vrednost čija vrednost 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.

```
const pi = 3.14
```

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 [**?**]:

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 broje označavamo i sa samo `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 **??**.

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 označavamo ih 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 u programskom jeziku Go, a to su `true` i `false`. Promenljive ili konstante koje čuvaju bulovsku vrednost su tipa `bool`.

Stringovi su nepromenljiva sekvenca bajtova. Stringovi 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 imamo string `s` deklarisan sa `s := "String "` i pokušamo da promenimo prvo slovo `s[0] = 'R'`, dobili bismo grešku od strane kompajlera. Ako bismo uradili konkatenaciju dva stringa operatorom `+` i uradili sledeće `s = s + "mutirani"`, ne bismo promenili prvobitni string `s`, već bi se napravio novi string `"String mutirani"` i dodelio promenljivoj `s`. [**?**]

4.2 Složeni tipovi podataka

U složene tipove podataka u programskom jeziku Go spadaju nizovi i strukture.

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...

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

Listing 5: Deklaracija nizova

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

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

Listing 6: Inicijalizacija niza prilikom deklaracije

Veličina niza je deo njegovog tipa, dakle niz `[3]Int` i niz `[4]Int` su različiti tipovi. Veličina mora biti konstantan izraz, čija vrednost može da se izračuna kada se program kompajlira. Sledeći izrazi daju grešku prilikom kompajliranja:

```
q := [3]int{1, 2, 3}
q = [4]int{1, 2, 3, 4} // compile error: cannot assign [4]int to
    [3]int
```

Listing 7: Primer greške 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 u programskom jeziku Go 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 ide ime novog tipa, pa ključna reč `struct` i onda definicija strukture unutar zagrada `{}`. Pogledajmo primer:

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

var golf Automobil;
```

Listing 8: Definicija strukture

Pojedinačnim poljima se pristupa pomoću tačka notacije. U primeru iznad, kada bismo želeli da saznamo snagu automobila `golf`, uradili bismo 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 referentne tipove podataka spadaju pokazivači, isečci, kanali, mape i funkcije.

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 možemo da pristupamo i menjamo vrednost promenljive bez znanja imena same promenljive. Ako je promenljiva deklarisana sa `var x int` onda operatorom `&` možemo dobiti adresu promenljive `x`.

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

Listing 9: Primer pokazivača

Vrednost na koju pokazuje neki pokazivač možemo dobiti operatorom `*`. U primeru iznad, vrednost na koju pokazuje `p` dobijamo sa `*p`. Takvu vrednost možemo i da izmenimo naredbom `*p = 2`. "Nula" vrednost za pokazivač na bilo koji tip je `nil`. Pokazivači su uporedivi, dva pokazivača su jednaka ako pokazuju na istu memorisku adresu ili ako su oba `nil`. Pokazivače možemo da kreiramo i bez promenljive, 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 tip *int , pokazuje na neku neimenovanu
     promenljivu
```

Listing 10: 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 (engl. chan) omogućavaju dvosmernu komunikaciju između dve gorutine. Oni će biti detaljnije opisani u poglavlju **??**.

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. Pomoću funkcije `make` možemo da kreiramo jednu mapu.

```
mapa = make(map[string]int) //Kljuc tipa string, a vrednost int
mapa["godine"] = 34
```

Listing 11: Kreiranje mape

Mapa se može kreirati i na drugi način, gde joj se prilikom kreiranja inicijalizuje vrednost:

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

Listing 12: Inicijalizacija mape prilikom deklaracije

Funkcije ćemo detaljnije predstaviti u poglavlju **??**.

4.4 Interfejsni tipovi podataka

Interfejsi izražavaju apstrakciju ili generalizaciju ponašanja nekog drugog tipa. Pomoću generalizacije, intefejsi nam dozvoljavaju da pišemo funkcije koje su fleksibilnije i adaptiranije zato što ne ulaze u detalje određene implementacije. O interfejsnim tipovima ćemo pisati u poglavlju **??**.

5 Kontrola toka

5.1 Naredbe grananja

U programskom jeziku Go razlikujemo dve naredbe grananja `if` odnosno `if else` naredba i `switch` naredba.

Naredbom `if` se provera neki uslov i u zavisnosti od istinitosne vrednosti tog uslova izvršava se jedan blok naredbi.

```
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` nam daje lepšu i čistiju sintaksu za ispitivanje većeg broja vrednosti.

```
switch broj {
case 1: // ...
case 2: // ...
case 3: // ...
case 4: // ...
default:
}
```

Listing 14: Primer `switch` naredbe

5.2 Petlje

Jedina petlja koja postoji u programskom jeziku Go je `for`. Ona ima sledeću sintaksu:

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

Listing 15: Primer `for` petlje

Za izlazak iz petlje možemo da koristimo naredbu `break`, a za prelazak na drugu iteraciju naredbu `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.

```
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 u programskom jeziku Go počinje sa ključnom reči `func`, zatim slede ime funkcije, lista parametara i lista povratnih vrednosti.

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

Listing 17: Deklaracija fukcije

Lista parametara predstavlja listu tipova i imena parametara i one se prosleđuju prilikom pozivanja funkcije. Povratne vrednosti predstavlja listu tipova povratnih vrednosti. Ako funkcija vraća jednu neimenovanu

vrednost ili ne vraća rezultat uopšte, onda lista povratnih vrednosti može biti izostavljena.

Pogledajmo primer:

```
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
```

Listing 19: Primer poziva funkcije

Kao što smo videli u primeru iznad, grupa parametara istog tipa se može grupisati, tako da jedan tip ne pišemo više puta.

Tip funkcije je ono što nazivamo potpis. Kažemo da 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 možemo imati više povratnih vrednosti za jednu funkciju.

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

Listing 20: Primer funkcije sa više povratnih vrednosti

Ovo je jako 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).

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. Pogledajmo primer ispod:

```
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)
}
// metoda 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 21: Primer metoda

5.4 Interfejsi

Interfejs izražava apstrakciju ili generalizaciju ponašanja nekog drugog tipa. Pomoću generalizacije intefejsi nam dozvoljavaju da pišemo funkcije koje su fleksibilnije i adaptiranije zato što ne ulaze u detalje određene implementacije.

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 imate neku vrednost nekog konkretnog tipa, tačno znate šta je ona i šta možete da „uradite“ sa njom.

Pored konkretnih tipova u Go-u postoje i interfejsi. Interfejs je apstraktan tip. Interfejs ne pokazuje reprezentaciju interne strukture svojih vrednosti ili osnovene operacije koje one podržavaju. Interfejs otkriva samo neke od svojih metoda. Kada imamo neku vrednost interfejsnog tipa, ne znamo ništa o tome šta je ona, znamo samo ono 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 22: Primer interfejsa

Moguće je da se jedan interfejs definiše kao kombinacija više drugih interfejsa, nešto kao nasleđivanje u Javi:

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

Listing 23: Primer interfejsa definisanog preko drugih interfejsa

U primeru iznad smo definisali interfejs `A`, kao kombinaciju interfejsa `B` i `C`. Dakle, neki korisnički tip može biti razmatran kao tip interfejsa `A`, ako može i kao tip interfejsa `B` i `C`.

6 Konkurentno programiranje

Pojam konkurentnog programiranja se odnosi na mogućnost da se delovi programa izvršavaju nezavisno jedni od drugih, što 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 koju želimo da izvršavamo konkurentno. Veći broj gorutina (M) se može preslikati u proizvoljan broj niti (N) operativnog sistema što predstavlja M:N model niti [**?**].

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.

```
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 24: Kreiranje kanala

7 Zaključak

U radu su predstavljene osnovne funkcionalnosti i mogućnosti jezika Go, koje predstavljaju osnovu za dalja istraživanja. Go nam 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.