UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

Elektronske lekcije o programskom jeziku Go

MASTER RAD

Student

Ana Batinić

Mentor

prof. dr Miroslav Marić

Beograd
2021.

Sadržaj

1 Uvod

Današnjica se ne može zamisliti bez interneta i dobre internet konekcije, budući da se na internetu nalazi velika količina podataka iz različitih oblasti, koja je dostupna neograničenom broju korisnika. Ne može se porediti način dobijanja tačnih informacija danas i nekada, kada su se te informacije istraživale u bibliotekama, arhivama i slično.

Zbog ubrzanog razvoja interneta i količine informacija koje su dostupne na njemu, internet je postao jedan od glavnih medijuma za prenošenje znanja i učenja. Iz svega toga nastao je veći broj veb platformi za učenje, od kojih je jedna eŠkola veba, razvijena na Matematičkom fakultetu i dostupna na adresi http://edusoft.matf.bg.ac.rs/eskola_veba/#/home. Jedna od prednosti ove platforme je u tome što su sve elektronske lekcije napisane na srpskom jeziku, čime se olakšava učenje veb tehnologija korisnicima iz Srbije i regiona. U ovom radu će biti opisane elektronske lekcije o programskom jeziku Go koje se nalaze u okviru eŠkole veba.

Go ili Golang je programski jezik opšte namene i otvorenog koda, nastao u kompaniji Google 2007. godine. Jezik su osmislili Robert Grisemer (Robert Griesemer), Rob Pajk (Rob Pike) i Ken Tompson (Ken Thompson). Ideja je bila osmišljavanje novog, jednostavnog i moćnog jezika kao alternativa programskom jeziku C++ za razvoj serverskih aplikacija. Programski jezik Go je imperativni, statički tipizirani, kompajlirani jezik. Sadrži automatsko upravljanje memorijom i funkcionalnosti za konkurentno programiranje. Go nije objektno-orijentisani jezik, ali preuzima neke od koncepata objektno-orijentisanih jezika, kao što su metodi i interfejsi. Prvenstveno je namenjen izradi serverskih aplikacija, kao i sistemskom programiranju. Zbog svoje jednostavnosti, čitkosti i velike brzine izvršavanja postao je veoma popularan u kratkom vremenskom periodu.

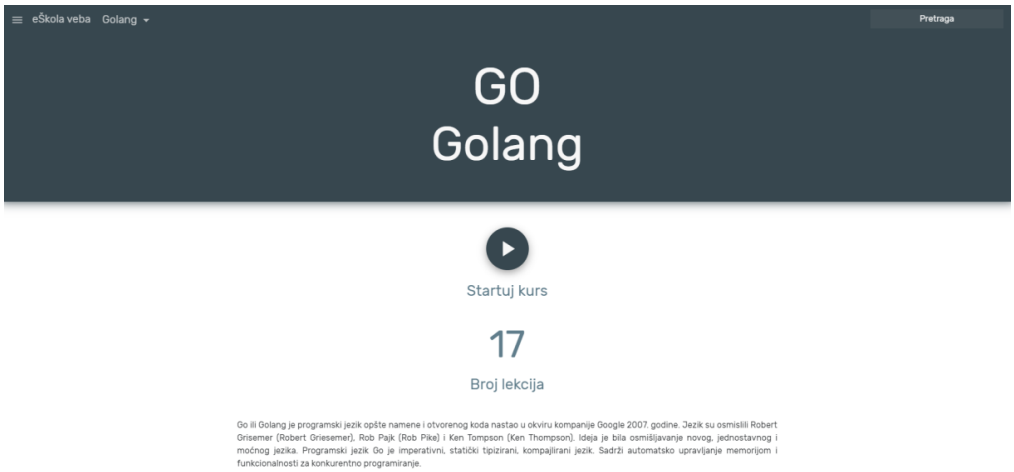
U ovom radu će biti prikazani osnovni pojmovi programskog jezika Go, kao i neke njegove naprednije funkcionalnosti. Na početku će biti opisana instalacija ovog programskog jezika, kao i unošenje, prevođenje i pokretanje programa. Zatim će biti reči o sintaksi ovog programskog jezika, kao i mogućnostima koje on pruža. Nakon toga biće obrađeni osnovni, kao i složeni tipovi podataka, grananja, petlje, funkcije i slično. Za korisnike koji savladaju osnovne pojmove programskog jezika Go biće dostupne naprednije teme, kao što su metodi, interfejsi, konkurentnost, paketi i testiranje.

2 Elektronske lekcije

Platforma eŠkola veba je elektronska platforma na kojoj se nalaze kursevi o popularnim veb tehnologijama. Svaki kurs je organizovan u vidu elektronskih lekcija koje su besplatne i svima dostupne. Prednost eŠkole veba u odnosu na druge platforme je u tome što su sve lekcije obrađene na srpskom jeziku. Više o platformi eŠkola veba može se videti u [1]. Izgled početne strane platforme prikazan je na slici 1.

Slika 1: Početna strana platforme eŠkola veba

Na platformi se nalaze kursevi o različitim veb tehnologijama, među kojima je i kurs o programskom jeziku Go koji je kreiran za potrebe ovog rada. Kurs je dostupan na adresi http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/go. Prilikom pokretanja kursa otvara se stranica prikazana na slici 2.



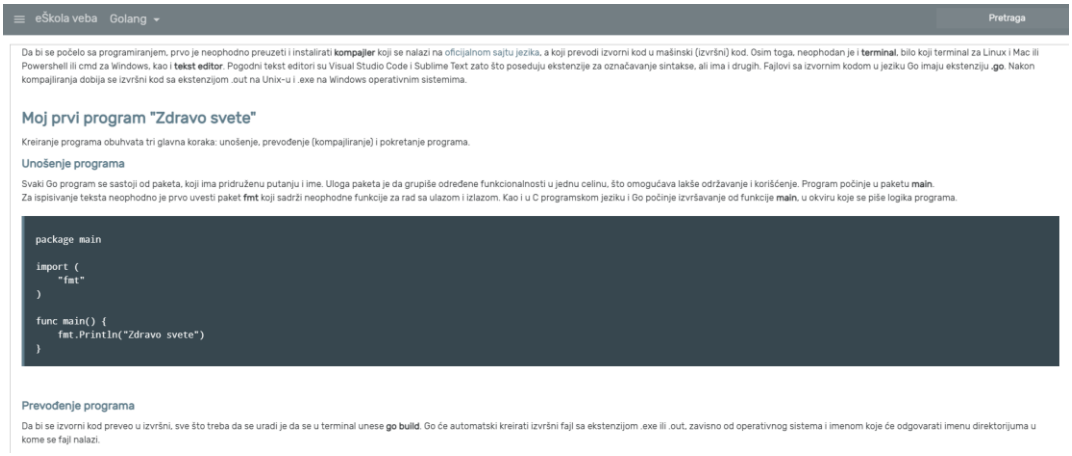
Slika 2: Naslovna strana kursa

Elektronske lekcije o programskom jeziku Go su kreirane da bi olakšale učenje ovog programskog jezika korisnicima koji se prvi put susreću sa njim, kao i da bi proširile znanje korisnika koji već imaju iskustva u radu sa programskim jezikom Go.

Za potrebe ovog rada kreirane su sledeće lekcije.

- Instalacija;
- Sintaksa programskog jezika Go;
- Promenljive;
- Tipovi podataka;
- Složeni tipovi podataka;
- Referentni tipovi podataka;
- Kontrola toka;
- Petlje;
- Funkcije;
- Metodi;
- Interfejsi;
- Konkurentnost;
- Paketi;
- Testiranje.

Izgled jedne od dostupnih lekcija prikazan je na slici 3.



Slika 3: Lekcija u okviru kursa

Lekcije su kreirane sistematično, odnosno prvo su objašnjeni osnovni pojmovi, a zatim se postepeno uvode složeniji. Svi koncepti i pojmovi su detaljno objašnjeni uz odgovarajuće primere pa na osnovu toga predstavljaju dobru osnovu za učenje programskog jezika Go.

3 Instalacija

Da bi se počelo sa radom u ovom programskom jeziku, prvo je neophodno preuzeti i instalirati *kompajler*, koji se nalazi na *oficijalnom sajtu programskog jezika Go*[1], a koji prevodi izvorni kod u mašinski (izvršni) kod. Osim toga, neophodan je i *terminal*, bilo koji terminal za Linux i Mac, Powershell ili cmd za Windows, kao i *tekst editor*. Pogodni tekst editori su Visual Studio Code i Sublime Text zato što poseduju ekstenzije za označavanje sintakse, ali ima i drugih. Fajlovi sa izvornim kodom u programskom jeziku Go imaju ekstenziju **.go**.

Nakon kompajliranja dobija se izvršni kod sa ekstenzijom .out na Unix-u i .exe na Windows operativnim sistemima.

Kreiranje programa obuhvata tri glavna koraka:

1. unošenje programa;
2. prevođenje (kompajliranje) programa;
3. pokretanje programa.

3.1 Unošenje programa

Svaki Go program se sastoji od *paketa*, koji ima pridruženu putanju i ime. Uloga paketa je da grupiše određene funkcionalnosti u jednu celinu, što omogućava lakše održavanje i korišćenje. Program počinje u paketu *main*.

Za ispisivanje teksta neophodno je prvo uvesti paket *fmt* koji sadrži neophodne funkcije za rad sa ulazom i izlazom.

Kao i u C programskom jeziku i Go počinje izvršavanje od funkcije **main**, u okviru koje se piše logika programa.

Primer 1: Prvi program koji ispisuje „Zdravo svete“

```
package main

import (
    "fmt"
)

func main() {
```

[1] https://golang.org/dl/

```
    fmt.Println("Zdravo svete")
}
```

3.2 Prevođenje programa

Da bi se izvorni kod preveo u izvršni, sve što treba da se uradi je da se u terminal unese *go build*. Go će automatski kreirati izvršni fajl sa ekstenzijom .exe ili .out zavisno od operativnog sistema i imenom koje će odgovarati imenu direktorijuma u kome se fajl nalazi.

3.3 Pokretanje programa

Da bi se novonastali program pokrenuo neophodno je u terminalu uneti imefajla.exe ili imefajla.out, gde je imefajla ime nastalo nakon komande go build. Drugi način pokretanja programa objedinjuje fazu prevođenja i izvršavanja. U terminal treba uneti *go run imefajla.go*, čime će se izvorni kod prevesti i pokrenuti, a rezultat izvršavanja prikazati.

4 Sintaksa programskog jezika Go

4.1 Identifikatori

Identifikatori predstavljaju imena koja se daju promenljivama, funkcijama, metodima, interfejsima, itd. Prilikom imenovanja treba pratiti jednostavna pravila, a to su: identifikatori počinju slovom ili donjom crtom i mogu imati bilo koji broj slova, cifara ili donjih crta.

Bitne stavke koje se moraju imati na umu su:

- identifikatori mogu da sadrže slova, brojeve ili _, ali ne smeju počinjati brojem;
- moraju biti jedinstveni, tj. jedan isti identifikator ne sme biti upotrebljen da označi dva entiteta;
- Go pravi razliku između malih i velikih slova;
- ne smeju sadržati prazan prostor;
- za identifikatore se ne smeju koristiti ključne reči koje su rezervisane u programskom jeziku Go.

4.2 Ključne reči

Ključne reči koje se ne smeju koristiti prilikom imenovanja identifikatora prikazane su u tabeli 1.

Tabela 1: Ključne reči

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

4.3 Komentari

Komentari su objašnjenja u kodu koje pišu programeri. Dodaju se sa ciljem da objasne neki deo koda i ignorisani su od strane kompajlera.

U programskom jeziku Go postoje dve vrste komentara.

1. Jednolinijski komentar,

```
// Ovo je komentar
```

2. Višelinijski komentar.

```
/*
  Ovaj komentar
  se prostire
  na vise linija
*/
```

5 Promenljive

Promenljiva je, po definiciji, imenovani prostor u memoriji koji skladišti neke podatke. U programskom jeziku Go se promenljive određenog tipa kreiraju korišćenjem ključne reči *var*, nakon čega sledi ime promenljive i njen tip, a potom se inicijalizuje nekom vrednošću.

Primer 2: Kreiranje promenljive

```
var ime tip = izraz
```

Dozvoljeno je izostaviti tip promenljive ili izraz, ali ne i oba. Ako se izostavi tip, onda se on zaključuje na osnovu vrednosti izraza. Ako se izostavi izraz, onda se vrednost inicijalizuje na

„nula vrednost“ određenog tipa. U slučaju numeričkih vrednosti to je *0*, za stringovske vrednosti to je *“”*, odnosno prazan string, za logičke vrednosti to je *false*, a za reference *nil*.

U programskom jeziku Go ne postoje neinicijalizovane promenljive.

Primer 3: Deklarisanje promenljivih sa i bez eksplicitne inicijalizacije

```
// promenljiva x je tipa int (ceo broj) i ima vrednost 1
var x = 1
// promenljiva y je takodje tipa int, ali je njena vrednost 0
var y int
```

Takođe, moguće je deklarisati i više promenljivih u jednoj liniji.

Primer 4: Deklarisanje više promenljivih u jednoj liniji

```
var x, y, z int // sve promenljive su tipa int
var x, y, z = true, false, 22 // promenljive x i y su tipa bool
                              // (istinitosna vrednost), a z je tipa int
```

5.1 Kratka deklaracija promenljivih

U okviru funkcija, moguće je koristiti kraću formu za deklaraciju i inicijalizaciju lokalnih promenljivih, oblika:

Primer 5: Kratka deklaracija

```
ime := izraz
```

Pošto tip promenljive nije naznačen u ovom obliku, tip promenljive je određen vrednošću izraza.

```
x := 0
y := 0.0
z := false
```

Zbog konciznosti **„kratka deklaracija“** je korišćeniji oblik, dok je deklaracija promenljivih korišćenjem ključne reči **var** najčešće korišćena za kreiranje promenljivih kojima se eksplicitno deklariše tip, a vrednost će biti naknadno dodeljena, a inicijalna vrednost nije važna.

I kod kratke deklaracije je moguće deklarisati više promenljivih u jednoj liniji.

Primer 6: Kratka deklaracija više promenljivih u jednoj liniji

```
x, y, z := 1.1, 2, "Zdravo svete"
```

5.2 Konstante

Moguće je deklarisati promenljive i korišćenjem ključne reči **const**. Međutim, u ovom slučaju se definiše promenljiva čija vrednost mora biti konstantna, tj. ne sme se menjati i mora biti poznata u fazi kompajliranja. Tip konstante mora biti jedan od „prostih“ tipova, odnosno numerička vrednost, logička vrednost ili string.

Primer 7: Deklaracija konstante

```
const pi = 3.14
const e = 2.71
```

5.3 Opseg promenljivih

Opseg promenljive je deo programa u kome se može pristupiti nekoj promenljivoj. U programskom jeziku Go postoje dve kategorije opsega, koje zavise od toga gde je promenljiva deklarisana. To su:

1. **lokalni opseg** (promenljive deklarisane u okviru bloka ili funkcije);
2. **globalni opseg** (promenljive deklarisane izvan funkcije).

Primer 8: Opseg promenljivih

```
package main

import "fmt"

var x int = 20 // globalna promenljiva


func prikazi_vrednost() {
    var y int = 10 // lokalna promenljiva

    // ovde je moguce prikazati vrednost promenljive y
    // zato sto je ona deklarisana u ovoj funkciji
    fmt.Printf("Vrednost lokalne promenljive je: %d\n", y)

}


func main() {

    // moguce je pristupiti globalnoj promenljivoj x
    // zato sto je deklarisana izvan funkcije main
    fmt.Printf("Vrednost globalne promenljive je: %d\n", x)
    // ali ne moze se prikazati vrednost promenljive y
    // zato sto je ona lokalna za funkciju prikazi_vrednost

}
```

6 Tipovi 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. Kao što je već rečeno, tip se ne mora eksplicitno navesti, već se može zaključiti na osnovu vrednosti koja joj je dodeljena kada se promenljiva deklariše.

6.1 Osnovni tipovi podataka

U osnovne tipove podataka u programskom jeziku Go spadaju: *numerički*, *stringovski* i *logički* tipovi.

Numerički tipovi pod sobom uključuju nekoliko vrsta celih brojeva, brojeva u pokretnom zarezu i kompleksnih brojeva.

6.1.1 Celi brojevi

U programskom jeziku Go postoji 11 vrsta celih brojeva, koji se razlikuju po memoriji koja je potrebna za njihovo čuvanje, kao i vrednostima koje mogu da čuvaju. Celi brojevi mogu da budu označeni i neoznačeni.

Označeni celi brojevi su:

- int8 – zauzima 8 bita;
- int16 – zauzima 16 bita;
- int32 – zauzima 32 bita;
- int64 – zauzima 64 bita.

Neoznačeni celi brojevi su:

- uint8;
- uint16;
- uint32;
- uint64.

Pored njih postoje i int, uint i uintptr, a najčešće se koriste int i uint, koji zauzimaju 32 ili 64 bita, zavisno od sistema.

Operacije nad celim brojevima prikazane su u tabeli 2.

Tabela 2: Operacije nad celim brojevima

+	Sabiranje	==	Jednakost
-	Oduzimanje	!=	Nejednakost
*	Množenje	>, >=, <, <=	Relacije veće, veće ili jednako, manje, manje ili jednako
/	Deljenje	&&	Logičko i
%	Ostatak pri deljenju	||	Logičko ili
++	Inkrementacija	!	Logička negacija
--	Dekrementacija	=	Operator dodele

Maksimalne vrednosti koje mogu da skladište promenljive tipa *int* lako je odrediti na osnovu broja bita koji stoji u tipu. Najveću vrednost koju uint8 može da čuva je 255, odnosno $2^8 - 1$, a najmanja je 0. Dok je za int8 najveća vrednost 127, odnosno $2^7 - 1$, a najmanja vrednost -128, odnosno -2^7.

6.1.2 Brojevi u pokretnom zarezu

U programskom jeziku Go postoje dva tipa brojeva u pokretnom zarezu i to su:

1. float32;
2. float64.

Zauzimaju 32 i 64 bita. Svi brojevi u pokretnom zarezu se skladište u memoriji po IEEE-754 formatu.

6.1.3 Kompleksni brojevi

Kompleksni brojevi su poslednji od numeričkih tipova i to su:

1. complex64;
2. complex128.

Zauzimaju 64 i 128 bita. Realni i imaginarni deo *complex64* su float32 vrednosti, a realni i imaginarni deo *complex128* su tipa float64.

Mogu se inicijalizovati na dva načina:

Primer 9: Inicijalizovanje kompleksnih brojeva

```
c1 := complex(7, 12) // koriscenjem funkcije complex
c2 := 7 + 12i // direktnom inicijalizacijom
```

Da bi se dobio realni i imaginarni deo kompleksnog broja koriste se funkcije *real* i *imag*.

Primer 10: Određivanje realnog i imaginarnog dela

```
c := complex(7, 12)
realniDeo := real(c)
imaginarniDeo := imag(c)
```

6.2 Logičke vrednosti

Postoje dve logičke vrednosti i to su *true* i *false*. Promenljive koje čuvaju logičke vrednosti imaju tip *bool*. Služe za predstavljanje istinitosne vrednosti. Operatori poređenja, kao što su >, <, ==, itd. se obično definišu tako da vraćaju logičke vrednosti.

Primer 11: Deklaracija bool promenljive

```
var grmljavina bool = true
```

6.3 Stringovi

Stringovi su nepromenljiva sekvenca bajtova i koriste se za predstavljanje teksta. Stringovi se kreiraju tako što se neki tekst stavi između dvostrukih znakova navoda (” i ”).

Primer 12: Deklaracija string promenljive

```
var ime = "Rada"
```

Stringovi su nepromenljivi, tako da se u slučaju promene nekog slova dobija greška od kompajlera. Podržavaju konkatanaciju operatorom +. Korišćenjem ovog operatora, ne menja se prvobitni string, već se kreira novi string.

Primer 13: Konkatanacija stringova

```
var s = "Rada"
s = s + " uci"
```

Što se kraće može zapisati korišćenjem operatorom += kao:

```
var s = "Rada"
s += " uci"
```

Stringovi se mogu porediti relacijskim operatorima >, <, >= i <=, pri čemu će se stringovi porediti leksikografski.

U tabeli 3 prikazani su specijalni znakovi.

Tabela 3: Specijalni znakovi

Literali	Opis
‘\n’	novi red (newline)
‘\r’	povratak na početak reda (carriage return)
‘\f’	sledeća strana (formfeed)
‘\b’	brisanje znaka levo od kursora (backspace)
‘\s’	razmak (space)
‘\t’	tabulator (tab)
‘\’’	jednostruki navodnik
‘\”’	dvostruki navodnik
‘\\’	kosa crta nalevo (backslash)
‘\ddd’	oktalni kod karaktera
‘\uxxx’	heksadekadni kod karaktera iz UNICODE skupa

Više informacija o osnovnim tipovima podataka se može pročitati u [2].

7 Složeni tipovi podataka

Složeni tipovi podataka su tipovi koji nastaju kombinacijom jednog ili više prostih tipova. U složene tipove podataka, u programskom jeziku Go, spadaju *nizovi* i *strukture*.

7.1 Niz

Niz je kolekcija promenljivih istog tipa, fiksne dužine od 0 ili više elemenata. Elementima niza se pristupa preko indeksa, koji se kreću od 0 do n-1, gde je n dužina niza. Kada se kreira niz u memoriji se rezerviše odgovarajući broj uzastopnih mesta koji odgovara dužini niza. U slučaju da u memoriji ne postoji dovoljno mesta dobija se greška.

Primer 14: Deklaracija niza i dodela vrednosti

```
package main

import (
    "fmt"
)

func main() {

    var x [10]int     // deklaracija niza od 10 elemenata tipa int
    x[1] = 10         // postavljanje vrednosti 10 na drugo mesto u nizu
    fmt.Println(x[1]) // ispisuje drugi element niza x
                      // posto indeksiranje krece od 0
}
```

Niz je moguće inicijalizovati prilikom njegove deklaracije.

Primer 15: Deklaracija i inicijalizacija niza

```
package main

import (
    "fmt"
)

func main() {

    var x [10]int = [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    // ili krace
    y := [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

}
```

Treba imati na umu da je veličina niza deo njegovog tipa, pa su [2]int i [3]int različiti tipovi. Veličina niza mora biti konstantan izraz, čija vrednost može da se izračuna prilikom kompajliranja programa. Dodeljivanjem nizu jednog tipa drugi tip dobija se greška.

Primer 16: Greška prilikom dodeljivanja niza

```
package main

import (
    "fmt"
)

func main() {

    var x [2]int = [2]int{1, 2}
    x = [3]int{1, 2, 3}
    // vraca gresku:
    // cannot use [3]int{...} (type [3]int) as type [2]int in assignment
    // posto su [2]int i [3]int razliciti tipovi

}
```

Nizovi se mogu porediti relacionim operatorima == i **!=**. Dva niza su jednaka ako imaju jednake vrednosti na istim indeksima.

Da bi se dobila dužina niza, koristi se funkcija *len()*.

Primer 17: Određivanje dužine niza

```
package main

import (
    "fmt"
```

```
)

func main() {

    var x [2]int = [2]int{1, 2}
    fmt.Println(len(x)) // vraca 2 kao sto je i ocekivano

}
```

7.2 Strukture

Struktura je složeni tip podataka u programskom jeziku Go, koji grupiše nula ili više podataka. Omogućava kreiranje složenih tipova podataka kojima se predstavljaju objekti iz svakodnevnog života. Svaki podatak ima svoje ime i vrednost. Struktura se kreira korišćenjem ključne reči *type*, nakon čega sledi ime novog tipa, pa ključna reč *struct* i onda definicija strukture unutar vitičastih zagrada **{}**.

Pojedinačnim poljima se pristupa koristeći . (tačku).

Primer 18: Kreiranje strukture

```
package main

type Osoba struct {
    ime string
    pol string
    godiste int
}

func main() {

    var ja Osoba;
    ja.ime = "Ana"

}
```

Moguće je i inicijalizovati promenljivu prilikom deklaracije.

Primer 19: Inicijalizacija strukture prilikom deklaracije

```
package main

type Osoba struct {
    ime string
    pol string
    godiste int
}

func main() {

    // eksplicitnim navodjenjem imena polja
```

```
    var ucenik1 Osoba = Osoba{ime: "Ana", pol: "Zenski", godiste: 1996}
    // ili krace bez navodjenja imena polja,
    // ali se mora postovati njihov redosled
    var ucenik2 Osoba = Osoba{"Jovana", "Zenski", 1996}
    // ili jos krace
    ucenik3 := Osoba{"Marko", "Muski", 1996}
}
```

Moguće je porediti pojedinačna polja strukture, kao i čitavu strukturu koristeći operatore == (jednako) i != (različito).

Primer 20: Poređenje struktura

```
package main

import "fmt"

type Tacka struct {
    x float32
    y float32
}

func main() {

    tacka1 := Tacka{1.35, 3.12}
    tacka2 := Tacka{1.35, 4.22}

    fmt.Println(tacka1.x == tacka2.x) // prikazuje true, posto su vrednosti
                                      // polja x jednake
    fmt.Println(tacka1 == tacka2) // prikazuje false, posto polja y imaju
                                  // razlicite vrednosti
}
```

8 Referentni tipovi podataka

Referentni tipovi podataka su tipovi koji čuvaju adresu ili referencu na vrednost. Kada se kreira promenljiva referentnog tipa rezerviše se prostor u memoriji koji čuva adresu drugog memorijskog bloka.

U referentne tipove podataka spadaju:

- pokazivači (pointers);
- isečci (slices);
- mape (maps);
- funkcije.

8.1 Pokazivači

Pokazivači su vrsta podataka koji čuvaju memorijsku adresu neke promenljive, tj. čuvaju lokaciju (adresu) gde se nalazi neki podatak.

Pomoću pokazivača može se pristupiti i menjati vrednost promenljive bez znanja imena te promenljive.

Operator **&** je unaran i naziva se operatorom referenciranja ili **adresnim operatorom** i vraća adresu svog operanda.

Primer 21: Kreiranje pokazivača

```
package main

import "fmt"

func main() {

    var x int = 1
    var p *int = &x // p je tipa pokazivac na int i pokazuje na promenljivu x
    fmt.Println(p) // prikazuje memorijsku adresu promenljive x

}
```

Vrednosti na koju pokazuje pokazivač pristupa se korišćenjem unarnog **operatora za dereferenciranje ***.

Primer 22: Pristupanje vrednosti na koju pokazuje pokazivač

```
package main

import "fmt"

func main() {


    x := 1
    fmt.Println(x)
    // prikazuje 1

    var p *int = &x
    *p = 10

    fmt.Println(x)
    // prikazuje 10

}
```

Promenom vrednosti operatorom dereferenciranja menja se vrednost i same promenljive, pošto se operatorom dereferenciranja modifikuje vrednost promenljive u memoriji. Nula vrednost za pokazivače je *nil*. Pokazivači se mogu porediti, a jednaki su ako pokazuju na istu memorijsku adresu ili ako su oba nil.

Moguće je kreirati i pokazivače bez promenljive korišćenjem funkcije *new()* koji alocira memoriju za novu promenljivu tog tipa i vraća adresu te promenljive.

Primer 23. Kreiranje pokazivača funkcijom new()

```
package main

import "fmt"

func main() {

    p := new(int) // p je tipa *int, tj. pokazivac na int
    *p = 10
    fmt.Println(*p)

}
```

Moguće je kreirati i pokazivače na strukture. Go dozvoljava pristupanje poljima strukture preko pokazivača bez korišćenja operatora za dereferenciranje, zbog kraće i čitljivije sintakse.

Primer 24: Kreiranje pokazivača na strukturu

```
package main

import "fmt"

type Tacka struct {
    X int
    Y int
}

func main() {

    tacka := Tacka{1, 2}

    var p *Tacka = &tacka
    fmt.Println(p.X) // sto je ekvivalentno sa (*p).X

}
```

8.2 Isečci

Isečci su sekvence elemenata promenljive dužine istog tipa. U programskom jeziku Go isečak se označava sa ***[]Tip***.

Izgledaju kao niz, ali se pri kreiranju ne navodi dužina. Za kreiranje isečaka se koristi funkcija *make()*.

Primer 25: Kreiranje isečka

```
package main

import "fmt"

func main() {
    s = make([]string, 2)
    s[0] = "ABC"
    s[1] = "DEF"
}
```

Isečci imaju tri komponente:

1. pokazivač;
2. dužina;
3. kapacitet.

Pokazivač pokazuje na prvi element u isečku.

Dužina je broj elemenata u isečku, tj. broj elemenata koji se zapravo nalazi u njemu i ne može biti veći od kapaciteta.

Kapacitet predstavlja broj elemenata za koji je rezervisan prostor u memoriji. Elementima se pristupa na isti način kao i u radu sa nizovima, a funkcija *len()* takođe služi za vraćanje dužine isečka. Isečci podržavaju još nekoliko funkcija koje ih čine „bogatijim“ od nizova. Jedna od tih funkcija je *append()* koja vraća isečak sa jednim ili više dodatih elemenata.

Primer 26: Dodavanje elementa u isečak

```
package main

import "fmt"

func main() {
    s = make([]string, 2)
    s[0] = "ABC"
    s[1] = "DEF"

    s = append(s, "GHI") // kreira novi isecak sa
                         // jos jednim dodatnim elementom

    fmt.Println(s[1])

}
```

Isečke je moguće kopirati korišćenjem funkcije *copy()*, a takođe podržavaju *„slice“* ([od:do]) operator, za uzimanje odsečka.

Primer 27: Kopiranje isečka

```
package main

import "fmt"

func main() {
    s = make([]string, 2)
    s[0] = "ABC"
    s[1] = "DEF"

    c := make([]string, len(s))
    copy(c, s) // kopiranje jednog isecka u drugi

    fmt.Println(c[0])

    l := s[0:1] // "slice" operator
    // uzima elemente od 0 do 1 iz isecka s
    // ukljucujuci element 0, ali bez elementa 1

    fmt.Println(l)

}
```

8.3 Mapa

Mapa u programskom jeziku Go označava heš tabelu, tj. neuređenu kolekciju sačinjenu od parova ključ-vrednost.

Mapa se označava sa *map[k]v*, gde je k tip ključa, a v tip vrednosti. Ključ mora biti jedinstven za svaku vrednost u heš tabeli.

Mapa se može kreirati na dva načina:

1. korišćenjem funkcije *make()*

```
mapa := make(map[string]string)
mapa["ime"] = "Ana"
mapa["pol"] = "zenski"
```

2. inicijalizacijom prilikom deklaracije

```
bodovi := map[string]int {"Ana": 100, "Jelena": 99 }
```

Kao nizovi i isečci, i mape podržavaju funkciju *len()* koja vraća broj elemenata mape. Dodatno, mape podržavaju funkciju *delete()* koja briše vrednost i ključ, na osnovu ključa koji joj se prosledi.

Primer 28: Određivanje dužine mape i brisanje vrednosti iz mape

```
package main

import "fmt"

func main() {

    bodovi := map[string]int { "Ana": 100, "Jelena": 99 }

    fmt.Println(len(bodovi))

    delete(bodovi, "Jelena")

    fmt.Println(bodovi) // prikazuje map[Ana:100]

}
```

Da bi se proverilo da li neka vrednost postoji u mapi mogu se uzeti dve vrednosti koje vraća pristup mapi nekim ključem. Prva vrednost predstavlja vrednost koja se nalazi u mapi za taj ključ, ako vrednost za taj ključ postoji, a druga vrednost je logička vrednost koja govori da li taj ključ postoji u mapi.

Ukoliko treba proveriti da li neki ključ postoji, a nije potrebna njegova vrednost, može se koristiti prazan identifikator (blank identifier), odnosno podvlaka (_).

Primer 29: Pristupanje elementima mape

```
package main

import "fmt"

func main() {

    bodovi := map[string]int { "Ana": 100 }

    _, postoji := bodovi["Jelena"] // _  se koristi kada vrednost
    // koja se nalazi na nekom kljucu nije bitna
    // vec se samo ispituje postojanje kljuca

    fmt.Println(postoji) // vraca false, posto kljuc nije prisutan u mapi

    osoba, postoji := bodovi["Ana"]

    fmt.Println(postoji, osoba) // vraca true 100
    // zato sto je kljuc Ana prisutan u mapi

}
```

9 Kontrola toka

Kontrola toka predstavlja redosled kojim se pojedinačni iskazi, instrukcije ili pozivi funkcija proveravaju ili izvršavaju.

9.1 Uslovne naredbe

Uslovne naredbe predstavljaju niz instrukcija koje se izvršavaju u slučaju da je zadovoljen specifičan uslov, odnosno ako je vrednost izraza logička vrednost *true*.

Programski jezik Go podržava dve vrste uslovnih naredbi, a to su *if-else* i *switch*.

9.1.1 If-else naredba

If naredba se koristi da bi se izvršila naredba ili blok naredbi u slučaju da je uslov ispunjen. Uslov je izraz koji vraća jednu od logičkih vrednosti *true* ili *false*. Slučaj **else** se izvršava ukoliko je vrednost koja se ispituje u if uslovu false.

Primer 30: If-else naredba

```
broj_bodova := 10
if broj_bodova > 5 {
    fmt.Println("Ispit polozen")
} else {
    fmt.Println("Ispit nije polozen")
}
```

Ukoliko postoji više slučajeva koji se ispituju, oni se mogu dodati proverom dodatnih uslova korišćenjem **else if**.

Primer 31: Pisanje dodatnih uslova korišćenjem else if

```
ocena := 5
if ocena == 5 {
    fmt.Println("Odlican uspeh")
} else if ocena == 4 {
    fmt.Println("Vrlo dobar uspeh")
} else if ocena == 3 {
    fmt.Println("Dobar uspeh")
} else if ocena == 2 {
    fmt.Println("Dovoljan uspeh")
} else {
    fmt.Println("Nedovoljan uspeh")
}
```

Ispitivanje se prekida kada se naiđe na prvi uslov čija je vrednost true ili kada se dođe do *else* uslova koji se ispunjava kada ni jedan od prethodnih uslova nije bio tačan. Moguće je izostaviti

else uslov, pri čemu ako se ne naiđe na istinit uslov, neće se izvršiti nijedna naredba, odnosno blok naredbi.

9.1.2 Switch naredba

Switch naredba omogućava upoređivanje početnog izraza ili vrednosti sa vrednostima koje su definisane u *switch bloku*. Ako se vrednosti poklapaju onda se izvršava niz iskaza definisanih u toj grani.

Za definisanje grane se koristi ključna reč *case*, nakon čega stoji skup naredbi koje treba da se izvrše. Za razliku od programskog jezika C i njemu sličnih jezika, u programskom jeziku Go nije neophodno navoditi ključnu reč *break* da bi se izašlo iz grane, pošto se nakon poslednjeg izraza u okviru grane izlazi iz switch iskaza. U slučaju da izvršavanje treba da pređe u blok sledećeg slučaja, može se iskoristiti ključna reč *fallthrough*.

Treba imati na umu da ključna reč *fallthrough*:

1. mora biti poslednji izraz u grani;
2. ne sme se nalaziti u poslednjoj grani.

Korišćenje *fallthrough*, *break* i *default* ključnih reči je opciono.

Primer 32: Upotreba switch naredbe uz ključne reči fallthrough, break i default

```
broj_godina := 3
switch broj_godina {
    case 1:
        fmt.Println("vrednost 1")
        break
    case 2:
        fmt.Println("vrednost 2")
    case 3:
        fmt.Println("vrednost 3")
        fallthrough
    default:
        fmt.Println("Podrazumevana vrednost")
}
```

10 Petlje

Programski jezik Go podržava samo jednu vrstu petlji i to je **for** petlja. *For* petlja je struktura koja se koristi za ponavljanje dela koda određeni broj puta.

U programskom jeziku Go ova petlja može biti zapisana na nekoliko načina.

- Kao obična for petlja koja ima sličnu strukturu kao for petlja u drugim programskim jezicima: C, C++, Java, C#, itd.

```
for inicijalizacija; uslov; korak {
    // telo petlje u okviru koga se pisu naredbe
}
```

  - *Inicijalizacija* je opciona i predstavlja izraz koji se izvršava samo jednom na početku petlje, najčešće je deklaracija promenljive, dodela ili poziv funkcije.
  - *Uslov* je izraz koji vraća logičke vrednosti true ili false i evaluira se na početku svake iteracije petlje. U slučaju da je uslov istinit petlja se izvršava.
  - *Korak* se izvršava nakon izvršavanja tela petlje. Nakon izvršavanja izraza koraka, ponovo se izračunava uslov, a u slučaju da je uslov false petlja se prekida.

Primer 33: Klasična for petlja

```
package main

import "fmt"

func main() {

    // for petlja
    // izvrsava se tri puta, dok vrednost promenljive i
    // ne dodje do 3 i uslov ne postane netacan, odnosno false
    for i := 0; i < 3; i++ {
        fmt.Println("Promenljiva i ima vrednost ", i)
    }

}
```

Rezultat izvršavanja primera 33.

```
Promenljiva i ima vrednost 0
Promenljiva i ima vrednost 1
Promenljiva i ima vrednost 2
```

- Drugi oblik korišćenja for petlje je „beskonačna“ petlja. Moguće je izostaviti sva tri elementa for petlje i zbog toga što nije naveden uslov izlaska iz petlje, smatraće se da je uslov istinit i petlja će se izvršavati beskonačno.

```
for {
    // naredbe
}
```

Primer 34: Beskonačna for petlja

```
package main

import "fmt"

func main() {

    for {
        fmt.Println("Zdravo svete")
    }

}
```

- Treći oblik korišćenja for petlje je kao *while* petlja u drugim programskim jezicima. U ovom slučaju se navodi samo uslov do kada se petlja izvršava.

```
for uslov {
    // naredbe
}
```

Primer 35: For petlja kao while petlja

```
package main

import "fmt"

func main() {

    i:= 0
    for i < 3 {
        i += 1
    }
    fmt.Println(i)
}
```

- Postoji još jedan oblik for petlje koji 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 {
    // naredbe
}
```

Primer 36: For petlja u kombinaciji sa naredbom range koja prikazuje indekse i vrednosti niza

```
package main

import "fmt"

func main() {

    studenti := []string { "Ana", "Jelena", "Nikola" }
```

```
    for indeks, vrednost := range studenti {
        fmt.Println(indeks, vrednost)
    }
}
```

Rezultat izvršavanja primera 36.

```
0 Ana
1 Jelena
2 Nikola
```

U nekim slučajevim treba preskočiti izvršavanje neke iteracije petlje, kao u slučaju prikazivanja samo parnih brojeva, tada se može koristiti ključna reč *continue*. Kada se naiđe na tu ključnu reč prelazi se na korak petlje i ponovo se ispituje njen uslov.

Primer 37: For petlja i naredba continue

```
package main

import "fmt"

func main() {

    var brojevi [10]int = [10]int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    for i := 0; i < len(brojevi); i++ {
        if brojevi[i] % 2 == 0 {
            fmt.Println(brojevi[i])
        } else {
            continue
        }
    }

}
```

Ukoliko je neophodno prekinuti izvršavanje petlje u nekom slučaju, pre nego što uslov postane netačan, može se koristiti ključna reč *break* koja prekida izvršavanje petlje kada se do nje dođe.

Primer 38: For petlja i naredba break

```
package main

import "fmt"

func main() {

    var brojevi [10]int = [10]int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    for i := 0; i < len(brojevi); i++ {
        if brojevi[i] != 5 {
            fmt.Println(brojevi[i])
```

```
        } else {
            break // izvrsavanje ove petlje ce se naglo prekinuti
                  // kada se dodje do broja 5.
        }
    }

}
```

11 Funkcije

Funkcija predstavlja logički organizovan deo koda koji je kreiran da bi se izvršavao neki specifičan zadatak.

Nakon što se definiše, funkcija se može pozvati više puta, što čuva memoriju i smanjuje redudantnost. U programskom jeziku Go funkcije predstavljaju referentni tip podataka.

Definicija funkcija u programskom jeziku Go počinje ključnom reči *func*, nakon čega sledi ime funkcije, lista parametara i lista povratnih vrednosti. Funkcija može da obavlja neki zadatak bez vraćanja vrednosti, a da bi se vratila neka vrednost iz funkcije koristi se ključna reč *return*.

Primer 39: Kreiranje funkcije

```
package main

import "fmt"

func povrsina_pravougaonika(duzina, sirina int) int {
    P := duzina * sirina
    return P
}

func main() {
    fmt.Println("Povrsina prvog pravougaonika je:", povrsina_pravougaonika(5,
7))

    fmt.Println("Povrsina drugog pravougaonika je:",
povrsina_pravougaonika(12, 15))
}
```

Lista parametara predstavlja listu tipova i imena parametara i one se prosleđuju prilikom pozivanja funkcije. Parametri istog tipa se mogu grupisati, tako da se jedan tip ne ponavlja više puta.

Za razliku od drugih programskih jezika, kao što su C, C++, Java, u programskom jeziku Go funkcije mogu vraćati više vrednosti[2].

Primer 40: Vraćanje više vrednosti iz funkcije

```
package main

import "fmt"

func zameni_vrednosti(a, b int) (int, int) {
    return b, a
}

func main() {
    a := 10
    b := 12

    a, b = zameni_vrednosti(a, b)

   fmt.Println(a, b)
}
```

Ovo je jako koristan koncept koji se koristi za upravljanje greškama i izuzecima. Prva vrednost može biti rezultat izračunavanja, a druga vrednost greška koja je nastala (ako je došlo do greške).

U programskom jeziku Go se koristi prosleđivanje parametara po vrednosti, tj. vrednosti koje se proslede funkciji se kopiraju u parametre funkcije. To znači da bilo kakva promena nad prosleđenim vrednostima u funkciji se ne odražava na vrednost sa kojom je funkcija pozvana.

Primer 41: Prosleđivanje parametara funkciji po vrednosti

```
package main

import "fmt"

func povecaj_vrednost(a int) {
    a += 1
}

func main() {
    a := 10

    fmt.Println("vrednost promenljive a je:", a)
    // prikazuje 10 kao sto je i ocekivano

    povecaj_vrednost(a)
```

[2] Mada je moguće simulirati ovo ponašanje vraćanjem strukture.

```
    fmt.Println("vrednost promenljive a je:", a)
    // i dalje se prikazuje 10, posto je vrednost promenljive a
    // kopirana i prosledjena funkciji
}
```

U slučaju da se želi promeniti vrednost promenljive koja se prosleđuje funkciji, može se koristiti prosleđivanje po adresi. Tada će parametar funkcije biti pokazivač, a funkciji će se prosleđivati adresa promenljive.

Primer 42: Prosleđivanje parametara funkciji po adresi

```
package main

import "fmt"

func povecaj_vrednost(a *int) {
    *a += 1
}


func main() {
    a := 10

    fmt.Println("vrednost promenljive a je:", a)
    // prikazuje 10 kao što je i ocekivano

    povecaj_vrednost(&a)
    // prosledjuje se adresa promenljive


    fmt.Println("vrednost promenljive a je:", a)
    // prikazuje 11 posto je promenjena vrednost koja se nalazi
    // na memorijskoj adresi
}
```

12 Metodi

U programskom jeziku Go **metodi** su funkcije koje se izvršavaju za specifične tipove. Sintaksa za definisanje metoda je slična kao i sintaksa za definisanje funkcija.

Razlika je u tome što metodi nakon ključne reči func sadrže dodatan parametar koji se zove *„receiver“* i koji govori za koji tip se metod vezuje.

Metodima nekog tipa se pristupa korišćenjem . (tačke).

Primer 43: Definisanje metoda

```
package main

import "fmt"

type Pravougaonik struct {
    sirina, visina int
}

func (p Pravougaonik) povrsina() int {
    return p.sirina * p.visina
}

func (p Pravougaonik) obim() int {
    return 2*p.sirina + 2*p.visina
}

func main() {
    p := Pravougaonik{sirina: 12, visina: 7}
    fmt.Println("Povrsina: ", p.povrsina())
    fmt.Println("Obim:", p.obim())
}
```

Metodi mogu da se definišu samo za tipove u istom paketu. Za ugrađene tipove kao što su int i string, ne mogu se definisati metodi, pošto oni pripadaju standardnom paketu *builtin*.

Kao i kod funkcija, i kod metoda se parametri prosleđuju po vrednosti, tj. sve promene nad „receiver“ parametrom se ne odražavaju na program izvan metoda.

Primer 44: Prosleđivanje parametara po vrednosti

```
package main

import "fmt"

type Pravougaonik struct {
    sirina, visina int
}

func (p Pravougaonik) PodesiSirinu (sirina int) {
    p.sirina = sirina
}

func main() {
    var p Pravougaonik // posto nije eksplicitno naznacena visina i sirina
                       // prilikom deklaracije, vrednost je 0
    p.PodesiSirinu(12)
    fmt.Println(p.sirina) // 0
}
```

Da bi se promene iz metoda odrazile na ostatak programa, neophodno je kao parametar metoda staviti pokazivač na tip.

Primer 45: Prosleđivanje parametara po referenci

```
package main

import "fmt"

type Pravougaonik struct {
    sirina int
    visina int
}

func (p *Pravougaonik) PodesiSirinu(sirina int) {
    p.sirina = sirina
}

func main() {
    var p Pravougaonik // posto nije eksplicitno naznacena visina i sirina
                       // prilikom deklaracije, vrednost je 0
    p.PodesiSirinu(12)
    fmt.Println(p.sirina) // 12
}
```

Pošto nije moguće kreirati metode za ugrađene tipove, već samo za tipove iz istog paketa, ukoliko je neophodno da se za osnovni tip dodaju neke metode može se kreirati novi tip korišćenjem "type alias", tj. kreiranjem drugog imena za postojeći tip.

Primer 46: Kreiranje type alias-a

```
package main

import "fmt"

type noviInt int

func (a noviInt) Saberi(b noviInt) {
    return a + b
}

func main() {
    broj1 := noviInt(10)
    broj2 := NoviInt(15)
    zbir := broj1.Saberi(broj2)
    fmt.Println("Zbir brojeva je", zbir)
}
```

Više informacija o metodima se može pročitati u [3].

13 Interfejsi

Interfejs je kolekcija potpisa metoda koje određeni tip može da implementira, tj. interfejs deklariše, ali ne definiše, ponašanje objekta određenog tipa. Osnovni posao interfejsa je da deklariše potpise metoda koji se sastoje od imena metoda, ulaznih parametara i povratnih vrednosti, dok je na konkretnom tipu da te metode definiše. Interfejs se kreira korišćenjem ključnih reči *type* i *interface*, između kojih stoji ime interfejsa, a nakon kojih idu potpisi metoda u vitičastim zagradama {}.

Na primer, poznato je da geometrijski oblici imaju površinu i obim, onda se može napraviti interfejs koji deklariše potpise metoda površina i obim.

Primer 47: Kreiranje interfejsa

```
package main

import "fmt"

type Oblik interface {
    Povrsina() float64
    Obim() float64
}

type Pravougaonik struct {
    sirina float64
    visina float64
}

func (p Pravougaonik) Povrsina() float64 {
    return p.sirina * p.visina
}

func (p Pravougaonik) Obim() float64 {
    return 2 * (p.sirina + p.visina)
}

func main() {
    var o Oblik
    o = Pravougaonik {7.0, 12.0}
    p := Pravougaonik {12, 15}
    fmt.Println(o.Povrsina())
    fmt.Println(p.Obim())
}
```

Tip koji sadrži metode koje određeni interfejs deklariše, odnosno ima iste potpise metoda, automatski implementira određeni interfejs.

Kada neki tip implementira interfejs onda se promenljiva tog tipa može prikazati tipom interfejsa, kao u prethodnom primeru.

U slučaju da određeni tip nema sve metode koje su deklarisane u interfejsu, a promenljivoj tipa interfejs bude pokušana dodela vrednosti tog tipa dolazi do greške.

Primer 48: Greške prilikom rada sa interfejsom

```
package main

import "fmt"

type Oblik inteface {
    Povrsina() float64
    Obim() float64
}

type Pravougaonik struct {
    sirina float64
    visina float64
}

func (p Pravougaonik) Obim() float64 {
    return 2 * (p.sirina + p.visina)
}

func main() {
    var o Oblik
    o = Pravougaonik {7.0, 12.0}
    // cannot use Pravougaonik literal (type Pravougaonik) as type Oblik in
assignment:
    // Pravougaonik does not implement Oblik (missing Povrsina method)
}
```

13.1 Prazan interfejs

U slučaju da interfejs ne sadrži metode, onda se naziva prazan interfejs i označava se sa *interface{}*. Pošto ovaj interfejs ne sadrži metode, svi tipovi implicitno implementiraju ovaj interfejs.

Ovo je jako koristan koncept koji se koristi u paketu *fmt*. Na primer, funkcija *Println* prihvata parametre koji su tipa interface{}.

Primer 49: Prazan interfejs

```
package main

import "fmt"

func prikazi(i interface{}) {
    fmt.Println("Vrednost je", i)
}

type Pravougaonik struct {
    sirina float64
```

```
    visina float64
}

func (p Pravougaonik) Povrsina() float64 {
    return p.sirina * p.visina)
}

func (p Pravougaonik) Obim() float64 {
    return 2 * (p.sirina + p.visina)
}

func main() {
    var p Pravougaonik = Pravougaonik {5, 7}
    prikazi(p)
}
```

Tip može da implementira više interfejsa, neophodno je samo da ima sve metode naznačene u interfejsima.

14 Konkurentnost

Konkurentnost je sposobnost programa da obavlja više stvari u približno isto vreme. Ovo znači da program može da obavlja više zadataka u približno isto vreme, ali da oni i dalje budu deo jednog programa.

Pojam konkurentnosti se često poistovećuje sa paralelizmom, ali paralelizam podrazumeva da se delovi jednog procesa izvršavaju istovremeno, dok konkurentnost definiše zadatke koji se izvršavaju u preklapajućim vremenskim intervalima, ali ne i istovremeno.

14.1 Niti

Niti su osnovne jedinice izvršavanja u okviru programa. Jedan proces može sadržati više niti koje se izvršavaju konkurentno.

Sve niti jednog procesa dele zajedničku memoriju i time štede memorijski prostor i vreme potrebno za kreiranje procesa.

Niti omogućavaju obavljanje drugih zadataka, dok su neki delovi blokirani ili zauzeti.

14.2 Gorutine

Gorutine u programskom jeziku Go označavaju funkciju koja se izvršava konkurentno. Gorutine su slične nitima, ali za razliku od njih gorutinama upravlja Go, a ne operativni sistem. Gorutine su karakteristične za jezik Go, a ime je nastalo spajanjem reči Go sa rečju korutina ili rutina. Kada se pokrene program, postoji samo jedna, glavna gorutina koja poziva funkciju main.

Gorutine se kreiraju korišćenjem ključne reči *go*, nakon čega sledi ime funkcije koja se pokreće konkurentno, tj. u novoj gorutini.

Primer 50: Kreiranje gorutina

```
package main

import (
    "fmt"
    "time"
)

func prikazi(zdravo string) {
    for i := 0; i < 3; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(zdravo)
    }
}

func main() {
    go prikazi("Zdravo")
    prikazi("svete")
}
```

14.3 Kanali

Kanali služe kao metod za sinhronizaciju gorutina. Kanali omogućavaju da gorutine dele memoriju.

Mogu se posmatrati kao *red* (queue) u okviru programa, gde jedne gorutine šalju podatke u kanal, a druge uzimaju podatke iz njega.

Korišćenjem operatora <- podaci se smeštaju u kanal ili preuzimaju iz njega, zavisno od toga sa koje se strane kanal nalazi.

Kanali se kreiraju korišćenjem funkcije *make()*, kojoj se kao parametar prosleđuje tip kanala. Tip kanala je određen tipom koji se želi čuvati.

Primer 51: Kreiranje kanala

```
package main

import "fmt"

func pomnozi_sa_2(c chan int, vrednost int) {
    c <- vrednost * 2
}
func main() {
    kanal := make(chan int)
    go pomnozi_sa_2(kanal, 5)
    go pomnozi_sa_2(kanal, 7)
```

```
    prva_vrednost := <-kanal
    druga_vrednost := <-kanal

    fmt.Println(prva_vrednost, druga_vrednost)
}
```

Kanali mogu biti jednosmerni i dvosmerni:

- **chan T** označava dvosmerni kanal tipa T. Kompajler dozvoljava uzimanje vrednosti iz kanala, kao i upisivanje vrednosti u njega.
- **chan<- T** označava send-only tip kanala. Kompajler ne dozvoljava uzimanje vrednosti iz kanala.
- **<-chan T** označava receive-only tip kanala. Kompajler ne dozvoljava slanje vrednosti u kanal.

Operacije nad kanalima:

1. zatvaranje kanala korišćenjem funkcije *close()*

```
close(ch)
```

2. slanje vrednosti u kanal korišćenjem operatora <-

```
ch <- vrednost
```

3. uzimanje vrednosti iz kanala, takođe korišćenjem operatora <-

```
vrednost := <-ch
```

4. određivanje kapaciteta kanala korišćenjem funkcije *cap()*

```
cap(ch)
```

5. određivanje dužine kanala korišćenjem funkcije *len()*

```
len(ch)
```

Više informacija o konkurentnosti se može pročitati u [4].

15 Paketi

Go programi su organizovani u pakete. **Paket** je kolekcija izvornih fajlova u istom direktorijumu koji se kompajliraju zajedno. Funkcije, tipovi, promenljive i konstante definisane u jednom izvornom fajlu su vidljive za ostale fajlove u okviru jednog paketa. Svaki Go program mora pripadati nekom paketu. Da bi se kreirao neki paket, koristi se ključna reč *package*.

Primer 52: Kreiranje paketa

```
package test

import "fmt"

func pozdrav(ime string) {
    fmt.Println("Zdravo", ime)
}
```

Deklaracija paketa mora biti prva linija koda u Go programu. Go programi počinju u paketu *main*, koji je poseban paket i koristi se za programe koji su namenjeni da se izvršavaju. Programi sa main paketom se nazivaju komande, a ostali se nazivaju paketi.

Postoje dva načina za importovanje paketa u programskom jeziku Go.

1. Korišćenje pojednačnih import naredbi,

```
import "fmt"
import "time"
import "math"
```

2. Grupisanje import naredbi.

```
import (
   "fmt"
   "time"
   "math"
)
```

Promenljive, tipovi ili funkcije koje počinju sa velikim slovom biće eksportovane i vidljive izvan paketa, dok su one koje počinju malim slovom vidljive samo u okviru istog paketa.

Primer 53: Vidljivost identifikatora u paketu

```
package main

import (
   "fmt"
```

```
    "math"
)

func main() {
    // MaxInt64 je vidljivo zato sto pocinje velikim slovom
    fmt.Println("Najveca vrednost int64 je: ", int64(math.MaxInt64))

    // kao i Phi
    fmt.Println("Vrednost Phi (ϕ) je: ", math.Phi)

    // pi pocinje malim slovom, pa se dobija greska od kompajlera
    fmt.Println("Vrednost Pi (π) je: ", math.pi)
    // cannot refer to unexported name math.pi undefined: math.pi
}
```

15.1 Moduli

Modul je kolekcija Go paketa u jednom direktorijumu u kome se nalazi **go.mod** fajl koji definiše način uvoženja paketa iz ovog modula.

Pre postojanja modula u programskom jeziku Go, svi paketi su morali da se nalaze u GOPATH-u, tj. direktorijumu koji se nalazi u GOPATH promenljivoj okruženja.

Da bi se kreirao modul, prvo je neophodno kreirati direktorijum, a onda otvoriti terminal i u tom direktorijumu uneti komandu:

```
go mod init lokacija/imeprojekta
```

U slučaju da se projekat nalazi na github-u, modul bi mogao biti kreiran kao:

```
go mod init github.com/imekorisnika/imeprojekta
```

Kada se pokrene Go program koji koristi pakete iz ovog modula, Go automatski preuzima taj modul sa naznačene putanje.

Primer 54: Korišćenje drugih paketa u modulu

```
package zdravo

import "rsc.io/quote"

func Zdravo() string {
    return quote.Hello()
}
```

Go će u go.mod dodati require rsc.io/quote v1.5.2, čime se naznačuje da ovaj modul koristi paket quote.

16 Testiranje

Testiranje je metoda provere da li program ispunjava određena očekivanja i vraća dobre rezultate. Svrha testiranja je da se otkriju greške i nedostaci pre prikazivanja programa korisnicima. Dobro testiran softver osigurava pouzdanost, bezbednost i dobre performanse.

16.1 Unit test

Unit test je funkcija koja testira određeni deo koda iz programa ili paketa. Njihov posao je da provere validnost aplikacije.

Da bi se počelo sa testiranjem prvo je neophodan direktorijum koji će skladištiti program i test fajlove, a on se može kreirati korišćenjem komande *mkdir*.

```
mkdir matematika
```

Korišćenjem komande *cd* se može pristupiti tom folderu.

```
cd matematika
```

U okviru foldera matematika, može se kreirati fajl operacije.go koji će sadržati osnovne matematičke operacije.

Primer 55: Kreiranje paketa sa nekim matematičkim funkcijama

```
package matematika

func Sabiranje(x, y int) (res int) {
    return x + y
}

func Oduzimanje(x, y int) (res int) {
    return x - y
}
```

U programskom jeziku Go ime test fajlova mora da se završava sa *_test.go* po konvenciji, a sve test funkcije moraju da imaju ime koje počinje sa *Test*, a prvo slovo sledeće reči mora biti veliko slovo. Za testiranje se koristi paket *testing*, i test funkcije u programskom jeziku Go primaju samo jedan parametar koji je pokazivač na *testing.T*.

Test koji testira prethodno kreirane matematičke operacije bi izgledao ovako:

Primer 56: Kreiranje testa za matematičke operacije

```
package matematika

import "testing"

func TestSabiranja(t *testing.T) {
    rezultat = Sabiranje(5, 7)
    ocekivan_rezultat = 12

    if rezultat != ocekivan_rezultat {
        t.Errorf("Ocekivana vrednost je %q, a funkcija je vratila %q",
rezultat, ocekivan_rezultat)
    }
}
```

U slučaju da test ne vraća odgovarajući rezultat koriste se *t.Error* ili *t.Fail* da bi se označilo da test nije prošao.

Da bi se test pokrenuo koristi se komanda *go test* koja pokreće fajlove koji imaju sufiks _test.go.

Rezultat ove funkcije može biti ***PASS***, u slučaju da je test prošao uspešno, ili ***FAIL*** u slučaju da test nije prošao.

17 Zaključak

Elektronske lekcije o programskom jeziku Go bi trebalo da olakšaju njegovo savladavanje i da zainteresuje čitaoce za ovaj jezik.

Programski jezik Go je za kratak vremenski period svog postojanja, zbog jednostavnosti, čitljivosti i velike brzine izvršavanja, uspeo da se pokaže kao dobra alternativa drugim programskim jezicima za razvoj veb aplikacija i sistemskom programiranju i da se ustali u IT zajednici.

U elektronskim lekcijama o programskom jeziku Go, koje se nalaze na platformi eŠkola veba, obrađene su, kako osnovne tako i neke naprednije funkcionalnosti ovog programskog jezika. Lekcije su kreirane sa ciljem da budu razumljive onima koji se prvi put susreću sa programiranjem kao i onima koji već imaju neko predznanje iz drugih programskih jezika. Svaka lekcija sadrži više primera koji upotpunjuju obrađenu temu i olakšavaju njeno razumevanje.

Sa savladanim pojmovima, obrađenim u elektronskim lekcijama o programskom jeziku Go, korisnik bi trebalo da bude osposobljen za rad u ovom programskom jeziku, kao i da ima dobru osnovu za proširivanje svog znanja, kako u programskom jeziku Go, tako i u drugim programskim jezicima.

Na internetu se može naći raznovrsna literatura o programskom jeziku Go koja je specijalizovana za određene oblasti programiranja u ovom programskom jeziku, kao što je veb programiranje ili pisanje desktop alata. Međutim, literatura koja se nalazi na internetu, najčešće je na engleskom jeziku, što može predstavljati prepreku korisnicima zbog nedovoljnog poznavanja stručnih izraza i pojmova.

Posebna prednost elektronskih lekcija o programskom jeziku Go je što su napisane na srpskom jeziku, i time olakšavaju razumevanje i učenje stručnih izraza i pojmova u ovom programskom jeziku. Elektronske lekcije su dostupne na platformi eŠkola veba na linku http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/go.

Literatura

[1] Jurić Nemanja, Marić Miroslav, *eŠkola Veba*, Matematički fakultet, Beograd, 2016.

[2] Brian W. Kernighan, Alan A.A. Donovan, The Go Programming Language, Addison-Wesley Professional, 2015.

[3] A tour of Go https://tour.golang.org/

[4] Mihalis Tsoukalos, Go od početnika do profesionalca, Kompjuter biblioteka, 2020.